В. А. Постернакова, Л. Ф. Пудовкіна, І. Б. Туркін

ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ТЕСТУВАННЯ

2019

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

В. А. Постернакова, Л. Ф. Пудовкіна, І. Б. Туркін

ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ТЕСТУВАННЯ

Навчальний посібник

Харків «ХАІ» 2019

УДК 004.89
П64

Рецензенти: д-р техн. наук, проф. В. І. Мойсеєнко,
канд. фіз.-мат. наук, доц. В. В. Веретельник

Постернакова, В. А.

П64 Якість програмного забезпечення та тестування [Електронний ресурс] : навч. посіб. / В. А. Постернакова, Л. Ф. Пудовкіна, І. Б. Туркін – Харків : Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харків. авіац. ін-т», 2019. – 63 с.

Розглянуто повний цикл верифікації і тестування програмного забезпечення з урахуванням зв'язку надійності його функціонування та якості програм. Показано особливості використання методів, засобів, видів тестування протягом технологічного циклу розроблення програмного забезпечення.

Для студентів, що спеціалізуються в галузі створення програмного забезпечення та використання сучасних інформаційних технологій для експлуатації в різних операційних системах.

Іл.10. Табл. 7. Бібліогр.: 12 назв

УДК 004.89

Вступ

На цей час треба створювати сучасне програмне забезпечення, яке обов'язково повинно бути конкурентоздатне, що визначається надійністю його функціонування і якістю складових функціонально закінчених програм.

Посібник містить теоретичний матеріал про різні види тестування та розроблення стратегій і методологій проектування верифікації та ефективної реалізації тестування.

Розглянуто повний цикл верифікації і тестування програмного забезпечення з урахуванням зв'язку надійності його функціонування та якості програм.

Приділено увагу особливостям верифікації і тестування програмного забезпечення, яке розробляється з використанням сучасних інформаційних технологій, для експлуатації в різних операційних системах.

Посібник призначений для студентів при виконанні лабораторних робіт і домашніх завдань, що створюють додатки різного призначення для експлуатації в різних операційних системах, але його також можна використовувати при виконанні курсових і дипломних проектів.

1 Розроблення плану системного тестування

1.1 Теоретичний матеріал

Системне тестування та способи його виконання.

1. Функціональне тестування – програмне забезпечення звіряється з зовнішньою специфікацією.
2. Тестування цілісності – звірення з документацією користувача.
3. Системне тестування – звірення з функціональними вимогами до ПЗ.

Існують такі п'ятнадцять категорій тестів для виконання системного тестування. Кожен з цих тестів доцільно застосовувати залежно від призначення ПЗ.

Тестування зручності використання.

Процедура перевірки полягає:

• у послідовному перегляді вихідного документа (речення за реченням);

• визначенні, чи виконує програма це завдання (результат потрібно відобразити на декількох рядках);

Таку перевірку можна здійснити без ПК.

Тестування на граничні обсяги – це виконання ПЗ на великих (конкретних) обсягах даних.

Тестування на граничних навантаженнях.

Мета тестування – показати, що ПЗ не може працювати при стресі й важких навантаженнях, а застосовується:

• в інтерактивних системах;

• в програмному забезпечені реального часу;

• при управлінні процесами (навантаження ОС максимально можливою кількістю завдань, граничним числом звернень до БД).

Тестування зручності експлуатації.

Зручність експлуатації пов'язано з психологічними проблемами та містить перевірку таких властивостей ПЗ системи:

а) чи можна пристосувати розроблений інтерфейс «ПК–користувач» для інформування та навчання кінцевого користувача;

б) чи витриманий глосарій у вихідних повідомленнях ПЗ;

в) чи зрозуміла діагностика помилок;

г) чи виявляється властивість «концептуальної цілісності ПЗ» (однаковість синтаксису, угод, семантик, формату, стилів скорочень);

д) чи містить система опції, кількість яких дуже велика, або використання малоймовірне;

е) чи видає система будь-які підтвердження на всі вхідні повідомлення;

ж) чи легко використовувати ПЗ.

Тестування захисту – це такі перевірки:

• чи здійснюється захист засобами ОС;

• чи використовується механізм захисту пам'яті ОС;

• який існує захист даних СУБД.

Тестування продуктивності.

Тестуються такі характеристики ПО:

а) час відгуку;

б) рівень пропускної здатності при певних навантаженнях і конфігурації;

в) переходи між режимами.

Тестування вимог до пам'яті.

При проектуванні деяких програмних продуктів визначаються граничні (конкретні) обсяги пам'яті та розміри тимчасових або буферних файлів.

Вимоги конфігурації обладнання.

Необхідно тестувати ПЗ при:

• мінімальній та максимальній конфігураціях;
• різних типах і кількостях пристроїв уведення-виведення;
• різних лініях зв'язку, протоколах;
• різному об’єму пам’яті.

Тестування сумісності.

Більшість програмних продуктів не є повністю новими, часто вони замінюють застарілі або недосконалі системи, тому з'являється необхідність забезпечити сумісність з існуючою системою і створити процедуру переходу від одного методу оброблення даних до іншого, сумісність і перетворення форматів даних.

Тестування зручності установки.

Контроль зручності настройки і складності процедури інсталяції.

Тестування надійності.

Обов'язково мають бути вказівки, що стосуються необхідної надійності експлуатації (час безперебійної роботи ПЗ, критерії надійності, умови експлуатації, граничні обсяги даних).

Тестується на багато користувачів робота в мережі.

Вимоги відновлення.

Контроль відповідності вимогам відновлення функцій відновлення.

Тестування зручності обслуговування – це перевірка зручності обслуговування:

• програмами діагностики та оброблення помилок;
• середній час на налагодження завдання.

Тестування документації.

Документація користувача має відповідати необхідній нормативній базі, погодженому з користувачем глосарію. Потрібно розробити всі необхідні документи.

Тестування процедур.

Виконання процедур користувачем потрібної кваліфікації (оператор ПЗ системи, адміністратор БД, користувач ПЗ системи).

1.2 План системного тестування

План системних випробувань являє собою документ, в якому узагальнено плани тестування окремих випадків використання і надання інформації про додаткові види тестування, які можна проводити на системному рівні.

Розділи цього плану наступні.

Вступ. У цьому розділі наведено посилання на застосовувані стандарти, а також планові документи на програмний продукт.

Тестовані елементи – програмні компоненти, що підлягають тестуванню: модулі, режими, фундаментальні функції, сервісні функції тощо, які треба перерахувати тут. Крім того, включені посилання на специфікації вимог і документацію до продукту (керівництво користувача, керівництво щодо установки продукту і т. д.).

Тестовані функції. Для них треба встановити відповідність з тестами.

Підхід. Це опис основних видів планованих робіт, які технології і засоби застосовуються для тестування кожної з основних груп функцій продукту, а також критерії системного **ТЕСТУВАННЯ** кожної групи. Треба навести базові вимоги, включаючи крайні терміни завершення робіт і вимоги до персоналу та наявності тестованих елементів.

Критерії проходження тестів, які визначають, пройшло чи ні тестований елемент ПЗ конкретний тест.

Документація. Список усіх тестових документів, які мають бути складені для специфікації даного продукту.

Завдання тестування. Усі завдання, які повинні бути вирішені під час підготовки до тестування і в процесі його проведення. Залежність між цими завданнями. Яка трудомісткість робіт при вирішенні цих завдань.

Необхідне обладнання. Необхідне для роботи апаратне і ПЗ, засоби тестування, обладнання тощо

Необхідний персонал. Фахівці, їхня кваліфікація, яка необхідна для вирішення поставлених завдань.

Календарний план. Перерахування ключових дат, термінів поставки необхідних ресурсів (людей, техніки, інструментальних засобів тощо).

Ризик і непередбачені обставини. Вказівка найгірших припущень, що стосуються виконання тестового плану, а також причини, які можуть вплинути на терміни виконання, і заходи, які будуть вжиті в цьому випадку.

Затвердження. Керівник, який затверджує тестовий план, підписи його і розробника.

Висновки.

2 Розроблення плану інтеграційного тестування

Тестування спільної роботи програмних модулів називається інтеграційним.

Завдання, які виникають у процесі тестування модулів:

- Планування тестування.
- Розроблення тестів.
- Формування налагоджувальних завдань.
- Власне тестування.
- Оброблення результатів тестування.

2.1 Методи виконання інтеграційного тестування

Методи виконання тестування можна розрізняти за такими ознаками.

- Форма подання модуля: символьний (на мові програмування), машинний код, об'єкти.
- Компоненти програми, на які спрямовано тестування: його структура або перетворення даних.
- Статичні або динамічні методи.

Інтеграційне тестування має найкращі показники ефективності та вартості.

Пропонуються такі методики тестування модулів:

- Тестування структури.
- Тестування оброблення даних.

Способи тестування взаємодії модулів. Існують два способи перевірки взаємодії модулів.

- Монолітне тестування.
- Покрокове тестування.

Нехай є ПЗ, що складається з декількох модулів, показаних на рисунку 2.1.

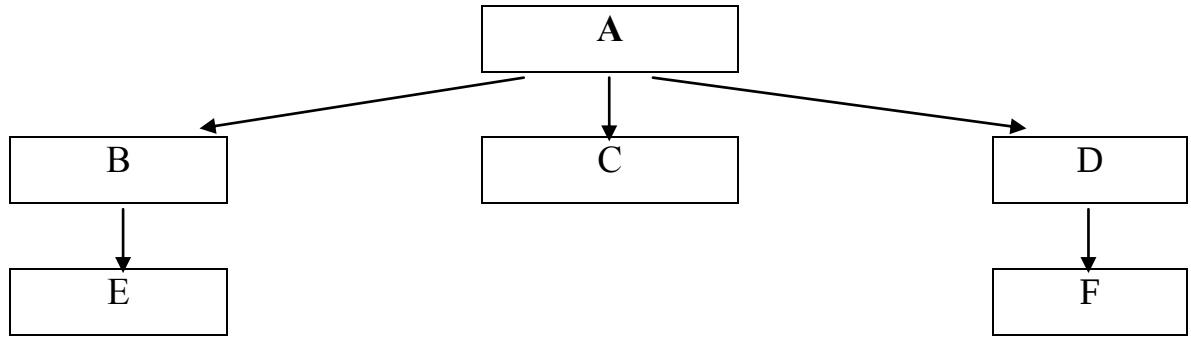

Рисунок 2.1 – Структура ПЗ

Монолітне тестування полягає у тому, що кожен модуль тестується окремо. Для кожного модуля пишеться один модуль-драйвер, який передає тестованому модулю управління, і один або кілька модулів-заглушок.

Наприклад, для модуля В потрібні дві заглушки, що імітують роботу модулів Е і А. Заглушка – це імітація виклику функції, що повертає ті ж дані, але більше нічого не виконує.

Коли всі модулі протестовані, вони збираються разом і тестується все ПЗ в цілому.

Покрокове тестування передбачає, що модулі тестуються не ізольовано, а підключаються по черзі до набору вже протестованих модулів.

Переваги та недоліки покрокового і монолітного тестувань наведено в таблиці 2.1.

Існує дві стратегії реалізації тестування:

• спадна, при якій тестування починається з верхнього головного модуля додатка;

• висхідна, що припускає тестування з термінальних модулів (що не викликають інших модулів).

Порівняльну характеристику цих стратегій дано в таблиці 2.2.

Таблиця 2.1 – Особливості реалізації тестування

Умови	Покрокове тестування по частинах	Монолітне тестування
Переваги	Локалізація помилок	Немає необхідності написання додаткового коду => дешевизна
Недоліки	Необхідність написання коду-оболонки (що викликає модуль) і коду-заглушки (імітує роботу іншого модуля)	Важко виявити джерело помилки Важко організувати виправлення помилок Погана автоматизація

Інтеграційні тести мають бути спрямовані на перевірку:

• основних компонентів коректної взаємодії;

• відповідність усіх даних, якими обмінюються основні компоненти через інтерфейс і які відповідають своїм специфікаціям;

• реалізація необхідних потоків управління.

ВИСНОВКИ.

На практиці вибір стратегії виконання тестування зазвичай вирішується просто: кожен модуль, по можливості, тестується відразу після його написання, у результаті чого послідовність тестування одних частин програмного забезпечення може відповідати висхідній стратегії, а інших – низхідній стратегії.

Виконання інтеграційного тестування часто замінюють модульним тестуванням.

Таблиця 2.2 – Особливості стратегій реалізації тестування

Переваги	Недоліки
Низхідна	
1. Чи має стратегія переваги, якщо помилки головним чином у верхній частині програми	Необхідно розробляти модулі-заглушки (часто дуже складні)
2. Подання тесту полегшується після підключення функцій вводу-виводу	Важко або неможливо створити тестові умови
3. Раннє формування структури додатка дозволяє зробити його показ користувачеві й служить моральним стимулом	Складно оцінити результати тестування
	Стимулюється до завершення тестування деяких модулів
Висхідна	
1. Має переваги, якщо помилки можливі головним чином в модулі нижнього рівня.	Необхідно розробляти модулі-драйвери
2. Легше створювати тестові умови	Додаток як єдине ціле не існує до тих пір, доки не доданий останній модуль
3. Простіше оцінити результат	

2.2 Модульне тестування ПЗ

Модульне тестування (Unit testing) – це процес перевірки коректності окремих модулів, класів, функцій програми.

Основна ідея - розроблення тестів, за допомогою яких можливо перевірити роботу кожної функції, методу класу, модуля.

Розроблення через тестування (test-driven development, TDD) – це техніка програмування, при якій тести для кожного модуля пишуться до (або паралельно) модулів додатка, тим самим керуючи їх розробленням.

2.2.1 Етапи розроблення в стилі Test Driven Development

Розроблення складається з коротких циклів (кроків), тривалість яких невелика.

Крок 1. **Витяг коду** зі сховища, де зберігаються і підтримуються у вигляді файлів будь-які дані (модулі, класи, функції, методи).

Розробник витягує вихідний код програмного забезпечення системи, що знаходиться в узгодженому стані, коли весь набір уже розроблених тестів для модулів виконується успішно.

Цей крок гарантує, що розробник має справу з останньою робочою версією вихідного коду проекту.

Крок 2. Додавання тесту. До існуючого набору тестів додається новий тест, який перевіряє, чи реалізує ПЗ системи деяку функцію або визначає деяку раніше невиявлену помилку.

Важливо написати тест до внесення змін у код модуля, тим самим ставлячи модулю нову вимогу.

Крок 3. Запуск тесту. Успішно виконується весь набір тестів, крім нового тесту, який виконується неуспішне.

Відбувається перевірка самого тесту, тобто чи включений тест у загальну систему тестування і чи правильно тест відображає нову вимогу до ПЗ системи, яку воно повинно задовольняти.

Крок 4. Виправлення помилки з мінімумом зусиль. Програмне забезпечення змінюється для того, щоб якомога швидше виконувалися всі тести шляхом самого простого рішення, що задовольняє новий тест. Одночасно не були внесені помилки в існуючі тести.

Велика частина небажаних побічних і віддалених ефектів від внесених у ПЗ змін відстежується саме на цьому етапі за допомогою досить повного набору тестів.

Крок 5. Запуск тестів. Після внесення виправлень у код модуля весь набір тестів повинен виконуватися успішно.

Якщо це не так, то необхідно:

- перевірити коректність нових вимог тесту;
- сумісність з попередніми вимогами;
- перевірити коректність внесених змін в код модуля.

Крок 6. Рефакторинг. Якщо необхідна в цьому циклі функціональність досягнута способом попереднього кроку, проводиться рефакторинг.

Процес зміни внутрішньої структури програми, що не зачіпає його зовнішньої поведінки і має на меті:

- полегшити розуміння його роботи;
- усунути дублювання.
- полегшити внесення змін у наступні версії

Крок 7. Запуск тестів. Рефакторинг пов'язаний із внесенням змін, які можуть випадково порушити роботу коду.

Необхідно домагатися успішного виконання всього набору тестів на даному етапі.

Крок 8. Фіксування змін. Перелік і докладний опис усіх змін, зроблених у цьому циклі в тестах і додатку, заносяться в репозиторій.

Тепер програмне забезпечення знову знаходиться в узгодженому стані й містить чітко відчутне покращення порівняно з попереднім станом.

Розроблення в стилі TDD – на рисунку 2.1.

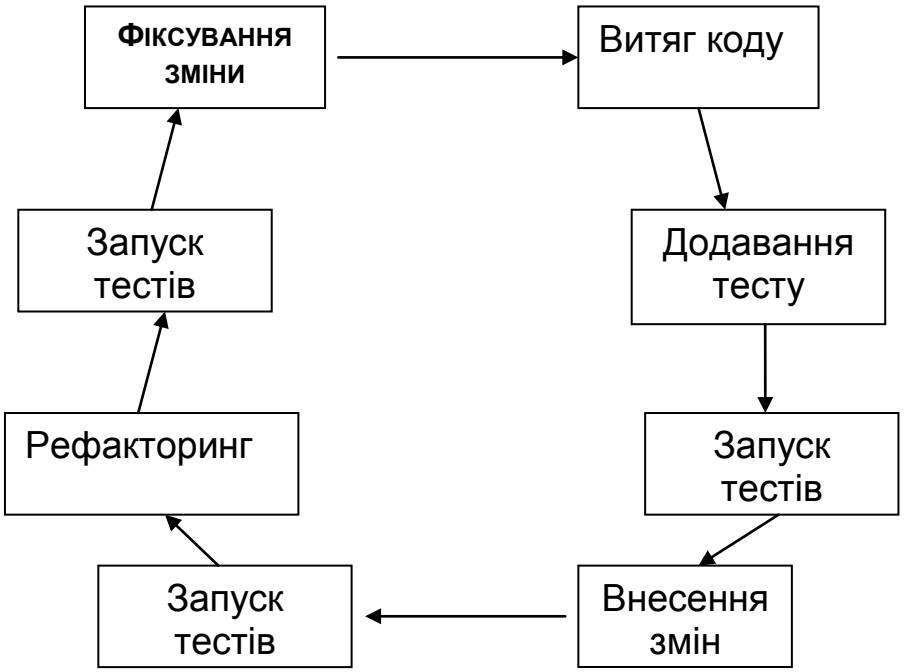

Рисунок 2.1 – Етапи розроблення в стилі TDD

2.2.2 Переваги використання Unit-тестування

Розроблені одного разу тести можна використовувати протягом життєвого циклу програми.

Полегшується спільна робота кількох людей над одними й тими ж компонентами системи.

Тести постійно керують процесом розроблення програми, ставляючи все нові й нові формальні вимоги до системи, і успішне проходження тестів свідчить про відповідність додатків заданим вимогам.

Полегшуються внесення змін і легкість виявлення помилок.

Спрощуються інтеграція окремих модулів і підвищення їх надійності.

Ретельно описані за шаблоном коди тестів можуть служити ілюстрацією використання класу, модуля, функції.

Спрощуються взаємозв'язки компонентів системи і підвищується ефективність контролю їх міжмодульних інтерфейсів.

Недоліки.

Попереднє створення якісних тестів збільшує календарний термін розроблення програми.

Протягом усього процесу створення програмного забезпечення системи треба витрачати час на контроль не тільки за актуальністю раніше написаних тестів, але і на необхідність зміни вимог до ПЗ системи.

2.2.3 Етапи додавання тестів до вже написаних класів.

1. Розроблення нових класів проводиться в стилі TDD, а старий код поки не змінюється. Немає шансу непомітно зламати вже працюючий код.

2. Зміна функціоналу або рефакторинг раніше створених модулів по можливості треба супроводжувати написанням тестів.

2.2.4 Що повинні покривати модульні тести?

В ідеалі – всі, але на практиці – критичні й нетривіальні ділянки коду, який:

- схильний до частих змін;
- від якого залежить працездатність великої кількості іншого коду;
- складний з великою кількістю залежностей.

2.2.5 Проблеми, що заважають упровадженню модульних тестів.

Побічні дії коду при інформаційному обміні з даними файлів і при виклику зовнішніх бібліотек.

1. Жорсткі залежності тестованого класу або велика кількість залежностей класу.

2. Виконання одним класом декількох обов'язків.

Усунення проблем № 1.

Винесення коду, що виконує побічні дії в окремі класи.

Зменшення залежностей між модулями (додавання рівнів непрямих).

Заміна конкретного інтерфейсу абстрактним.

Усунення проблем № 2

Розбиття класу на більш дрібні, кожен з яких несе відповідальність тільки за одну функціональність.

Створення бібліотек фальшивих об'єктів для тестування класів з великою кількістю залежностей.

ВИСНОВКИ.

Код, покритий тестами, має більшу надійність функціональності й більш високу якість складових компонент.

Розроблення тестів потребує додаткового часу розроблення, що треба враховувати при складанні попередньої вартості.

Відсутність тестів призводить до ускладнення підтримки розробленого коду і до збільшення витрат на проект і термінів розроблення.

Тестування коду, перш за все, – це завдання програміста, автора вихідного коду, а вже потім – тестувальника. Ефективність проектування і виконання тестування приходить з досвідом.

3 Методи «чорного ящика»

При використанні методології тестування за принципом «чорного ящика» передбачається, що код програми невідомий.

До тестування за принципом «чорного ящика» належать такі методи:

1. Метод застосування функціональних діаграм.
2. Метод виділення класів еквівалентності.
3. Метод аналізу граничних значень вхідних даних.
4. Метод припущення помилки.

3.1 Метод функціональних діаграм

Метод функціональних діаграм (або діаграм причинно-наслідкових зав’язків) дозволяє перевіряти вихідні специфікації програм на неповноту і неоднозначність.

Функціональна діаграма являє собою формальну мову, на якій транслюється специфікація, написана на природній мові.

Метод функціональних діаграм оснований на аналізі семантичного змісту зовнішніх специфікацій і переведення їх на мову логічних відносин між вхідними даними (ситуаціями) і вихідними даними і перетвореннями (ефектами).

Побудова тестів за допомогою цього методу здійснюється в кілька етапів.

1. Специфікація розбивається на «робочі» ділянки. Це пов'язано з тим, що функціональні діаграми стають занадто громіздкими при використанні цього методу в роботі з великими специфікаціями.

2. У специфікації визначаються ситуації (причини) і ефекти (слідства).

Ситуація є окремою вхідною умовою або такою, що відповідна класам еквівалентності вхідних умов.

Ефект – вихідна умова або перетворення системи (залишкова дія, якій вхідна умова передає стан програми або системи).

Ситуації і ефекти визначаються шляхом послідовного (слово за словом) читання специфікації. При цьому виділяються слова або фрази, які описують ситуації і ефекти. Кожній ситуації і ефекту присвоюється певний номер.

3. На основі аналізу семантичного змісту специфікації будується булев граф, що зв'язує ситуації і ефекти. Булев граф і є функціональною діаграмою.

4. Діаграма супроводжується примітками, за допомогою яких задають обмеження і описують комбінації ситуацій та (або) ефектів, які є неможливими через синтаксичні або зовнішні обмеження.

5. За допомогою методичного простеження поточного утримання умов діаграма перетворюється в таблицю рішень з обмеженими входами. Кожен стовпець таблиці відповідає тесту.

6. Стовпці таблиці рішень перетворюються в тести.

Базові символи для запису функціональних діаграм показано на рисунку 3.1. Кожен вузол діаграми може існувати в двох станах 0 або 1; 0 позначає стан «відсутній», а 1– стан «присутній».

Функція **ТОТОЖНІСТЬ** установлює: якщо значення дорівнює 1, то і значення є 1; в іншому випадку значення – 0.

Функція **НЕ** позначає: якщо значення дорівнює 1, то значення – 0; інакше дорівнює 1.

Функція **АБО** позначає: якщо значення 1, або дорівнює 1, тобто 1; в іншому випадку – 0.

Функція **І** позначає: якщо всі значення 1, і дорівнює 1; в іншому випадку – 0. Останні дві функції дозволяють мати будь-яке число входів.

Проілюструємо використання функціональної діаграми для отримання тестів. Для цього слід скористатися специфікацією, зображеною на рисунку 3.1.

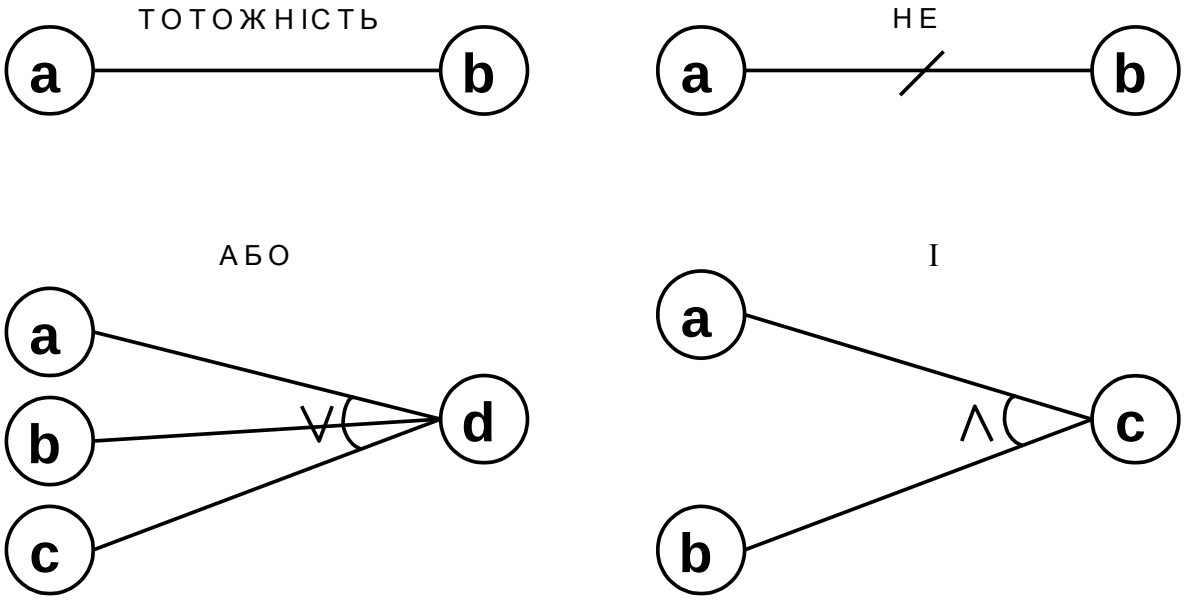

Рисунок 3.1– Базові символи для побудови функціональної діаграми

На **першому етапі** виділимо умови на вході – ситуації і довільно пронумеруємо їх:

Ситуація № 1. Перший, відмінний від пробілу символ, наступний за символом «С» і одним або декількома пропусками, є символ «\».

Ситуація № 2. Команда містить рівно два символи "\\".

Ситуація № 3. Довжина рядка 1 дорівнює одиниці.

Ситуація № 4. Довжина рядка 1 дорівнює 30.

Ситуація № 5. Довжина рядка 1 знаходиться між 2 і 29.

Ситуація № 6. Довжина рядка 2 дорівнює нулю.

Ситуація № 7. Довжина рядка 2 дорівнює 30.

Ситуація № 8. Довжина рядка 2 знаходиться між 1 і 29.

Ситуація № 9. Поточний рядок містить входження рядка 1.

На **другому етапі** виділимо такі ефекти:

Ефект 31. Надруковано змінений рядок.

Ефект 32. Перше входження рядка 1 у розглянутий рядок замінено на рядок 2.

Ефект 33. Надруковано «Не знайдено».

Ефект 34. Надруковано «Синтаксична помилка».

Проаналізувавши ситуацію і ефекти, можна зробити висновок: ефект № 32 означає, що в специфікаціях уже виявлені неоднозначності; явно не сказано, що відбувається, коли є кілька входжень рядка 1 у розглянутий рядок. Варто було б указати, що змінюється тільки перше входження.

На **третьому етапі** будуємо функціональну діаграму. Ситуації у вигляді вершин зображуємо на лівому краї, а ефекти – на правому. За допомогою базових логічних відносин, зображених на рисунку 3.1, і з ситуацій і ефектів будуємо структуру логічних відносин (булев граф). На рисунку 3.2. показано функціональну діаграму заданої специфікації.

Приклад. Команда «**ЗМІНИТИ».**

Призначення команди: використовується для зміни послідовності символів у поточному рядку редагованого файлу.

Синтаксис команди **ЗМІНИТИ :**

ЗМІНИТИ \ РЯДОК 1 \ РЯДОК 2

Після імені команди «**ЗМІНИТИ**» повинен бути, принаймні, один пробіл.

Вхідні дані.

Рядок 1 – це рядок символів, який Ви хочете замінити. Він може мати довжину від 1 до 30 символів і складатися з будь-яких літер, за винятком «\».

Рядок 2 – рядок символів, який повинен замінити рядок 1. Він може мати довжину від 0 до 30 символів і складатися з будь-яких літер, за винятком «\». Якщо рядок 2 пропущений (має нульову довжину), рядок 1 просто викреслюється.

ВИСНОВКИ.

1. Змінений рядок при вдалому завершенні команди виводиться на екран. Якщо зміну не можна бути виконати, оскільки не вдається знайти в поточному рядку рядок 1, друкується повідомлення «Не знайдено». Якщо синтаксис команди помилковий, друкується повідомлення «Синтаксична помилка».

2. Якщо синтаксис правильний і рядок 1 знайдений у поточному рядку, тоді рядок 1 видаляється, а його місце займає рядок 2. При

цьому поточний рядок розширюється або стискається залежно від різниці довжин рядка 1 і рядка 2. Якщо команда синтаксично неправильна або в поточному рядку не вдається знайти рядок 1, поточний рядок залишається незмінним.

На **четвертому етапі** складаємо таблицю рішень. Для цього послідовно вибираємо ефекти і записуємо всі комбінації ситуацій, які їх викликають. Потім виписуємо також стан усіх інших ефектів при цих комбінаціях. Так, наприклад, перші дев'ять стовпців у таблиці це комбінації ситуацій, що викликають ефект 31.

На **п'ятому етапі** перетворимо кожен стовпець таблиці рішень в окремий тест. Таким чином, у розглянутому прикладі загальна кількість тестів, які необхідно виконати, дорівнює 16.

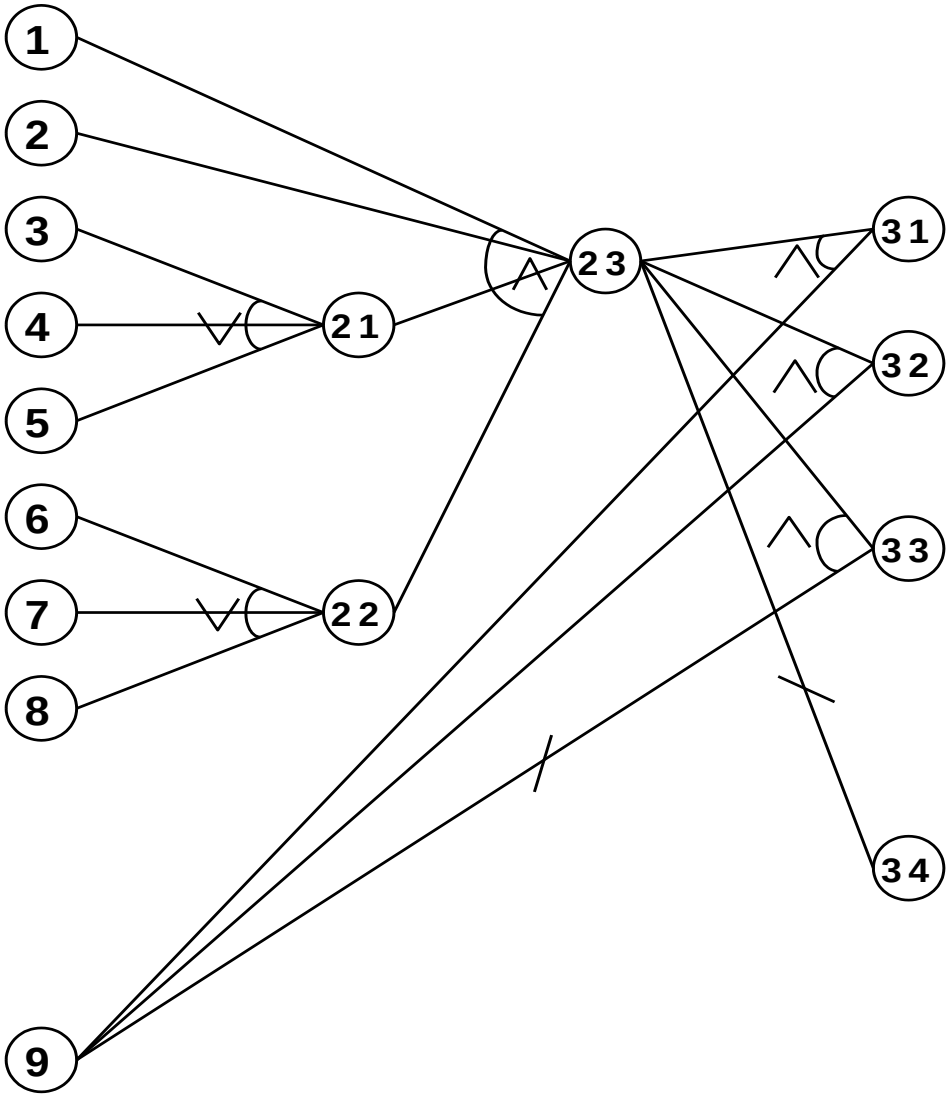

Рисунок 3.2 – Функціональна діаграма

Наведемо тест, що відповідає першому стовпцю таблиці рішень.

Тест № 1

На вхід програми подаються	
Поточний рядок	АВЕВАААААААААААААААААААААА
Команда	ЗМІНИТИ \Е\В
Очікуваний результат на виході програми	
Поточний рядок	АВВАААААААААААААААААААААА

Наведемо тест, що відповідає другій колонці таблиці рішень.

Тест № 2

На вхід програми подаються	
Поточний рядок	АВЕВАААААААААААААААААААААА
Команда	ЗМІНИТИ \ъ\ь
Очікуваний результат на виході програми	
Повідомлення	«Не знайдено»

ВИСНОВКИ.

1. За допомогою функціональних діаграм можна визначити такі помилки в специфікації:

• наявність ситуацій або комбінацій ситуацій, які не мають видимих ефектів;

• наявність ефектів, не пов'язаних з ситуаціями на вході.

2. За допомогою таблиць рішень можна виконувати такі дії:

- визначити відсутні ефекти;
- мінімізувати кількість тестів;
- передбачити очікуваний результат;
- виявити неоднозначність специфікації.

3.2 Класи еквівалентності даних

При розробленні тестів може виникнути ситуація, коли різні вхідні дані призводять до одних і тих же реакцій системи. Якщо при цьому такі вхідні значення мають щось спільне, то їх можна об'єднати в класи еквівалентності (КЕ), тобто виконати еквівалентне розбиття множини допустимих вхідних значень.

Розбивка на класи еквівалентності зменшує необхідну кількість тестів. Якщо в тестових вимогах не оговорено інше, при тестуванні досить виконати тільки один тест для кожного класу еквівалентності. Розбивка на КЕ особливо корисна, коли на вхід системи можна подати велику кількість різних значень; тестування кожного можливого значення призвело б до дуже значного обсягу тестування.

Розглянуті вище граничні умови можуть бути прикладом класів еквівалентності:

- значення з середини інтервалу;
- граничні значення;
- неприпустимі значення за межами інтервалу.

Таким чином, тестування граничних умов є окремими випадками тестування з використанням класів еквівалентності: замість того щоб тестувати усі неприпустимі значення, вибираються тільки суміжні з граничними значеннями.

При визначенні класів еквівалентності слід керуватися правилами:

- завжди має бути щонайменше два класи: коректний і некоректний;
- якщо за вхідною умовою визначається діапазон значень, то зазвичай є три класи: менше нижньої межі діапазону, всередині діапазону і більше верхньої межі діапазону (значення на кінцях діапазону можна вважати граничними значеннями);
- якщо елементи діапазону обробляються по-різному, то кожному варіанту оброблення будуть відповідати різним вимогам.

Іншим прикладом розбиття на класи еквівалентності є тестування відкриття файлу за його іменем. У результаті тестування має бути визначено, що усі варіанти імен оброблено системою відповідно до таких тестових вимог:

1) система виводити повідомлення про помилку при наявності в імені файла символів, які не є латинськими літерами або цифрами;

2) система виводити повідомлення про помилку, коли з таким ім'ям файл існує;

3) система має бути незалежною від регістрів символів імені файлу;

4) система має відкривати файл з іменем, який не суперечить вимогам 1 – 3.

Приклад. Вхідними значеннями тестів є різні імена файлів, вихідні - реакція системи (помилка або успішне відкриття).

Можна виділити такі класи еквівалентності:

а) по довжині імені (L) {L <16, L = 16, L> 16};

б) по типу символів:

- ім'я, що складається з цифр і букв будь-якого регістра;
- ім'я, що складається з цифр і букв нижнього регістра;
- ім'я, що складається з цифр і букв верхнього регістра;
- ім'я, що складається тільки з цифр;
- ім'я, що складається тільки з букв;
- ім'я, що містить керуючі символи.

ВИСНОВКИ**.**

Ці класи еквівалентності ілюструють, що перевірки на кордонах інтервалів можна застосовувати і для тестування арифметичних операцій і операцій порівняння.

Майже для всіх даних можна визначити «мінімальні» і «максимальні» допустимі значення.

3.3 Метод аналізу граничних значень вхідних даних

Граничні умови – це ситуації, що виникають безпосередньо вище або нижче межі вхідних або вихідних класів еквівалентності.

Загальні правила для проведення тестів.

1. Будують набори даних тестів для кордонів області з неправильними вхідними даними, і даних, які незначно виходять за межі області.
2. Будують набори даних тестів:
   - для мінімального і максимального значень;
   - тести більше і менше цих значень;
   - для перевірки всіх умов.
3. Використовують правила 1 і 2 для кожної вихідної умови.
4. Будують набори даних тестів для критичних граничних умов.

3.4 Метод припущення помилки

Використовувати цей метод може фахівець, що має практичний досвід, тобто підсвідомо він передбачає ймовірні типи помилок і потім розробляє тести для їх виявлення.

4 Розроблення програмного забезпечення з генерації набору даних для тесту

Цей розділ присвячено розробленню ПЗ для генерації набору даних для окремих тестів. Це підхід, при якому спочатку розробляється сценарій тестування, а потім пишеться і тестується програмний код. Такий підхід застосовується доти, доки не будуть вичерпані об'єкти тестування.

При розробленні тестів (та генерації набору даних) використовується інформація від різних розробників продуктів, включаючи реалізації варіантів, моделі проекту та інтерфейси класифікаторів. Тести виконуються після створення компонентів.

Як правило, проекти тестів створюються безпосередньо перед виконанням тестів і через значний час з моменту створення робочих продуктів проекту програмного забезпечення.

Приклад такого підходу показано на рисунку 4.1. У даному випадку розроблення тестів починається ближче до кінця етапу реалізації. У першу чергу при розробленні використовуються результати проектування компонентів. Стрілка від реалізації до

ВИКОНАННЯ ТЕСТІВ ГОВОРИТЬ ПРО ТЕ, ЩО ВИКОНУВАТИ ТЕСТИ МОЖНА ТІЛЬКИ ПІСЛЯ ЗАВЕРШЕННЯ РЕАЛІЗАЦІЇ.

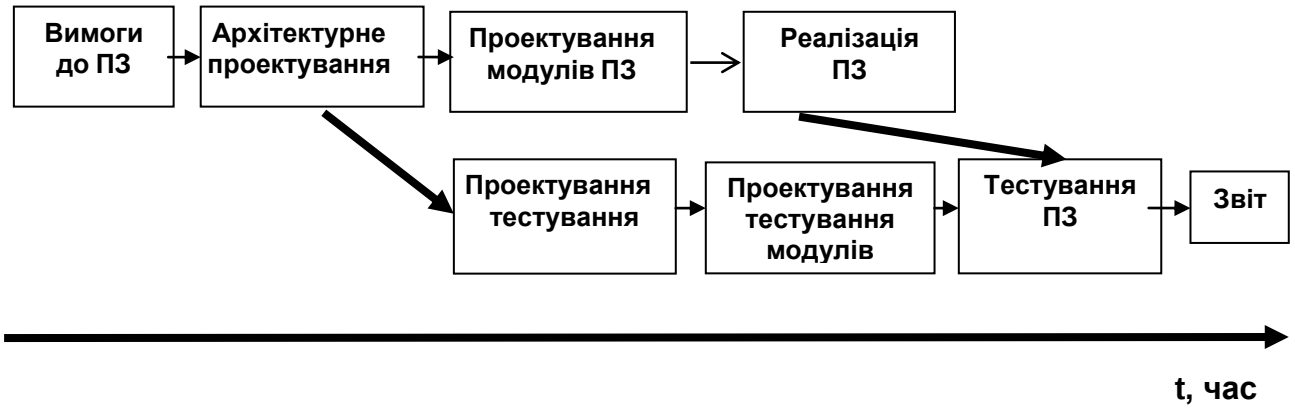

Рисунок 4.1 – Етапи проектування та виконання тестів

Проте це не єдина можлива організація. Хоча виконати тести можна тільки після реалізації компонентів, створити тести можна і до цього. Наприклад, тести можна створити відразу ж після створення робочих продуктів проектування. Більше того, тести можна створювати одночасно з проектуванням компонентів, як показано на рисунку 4.2.

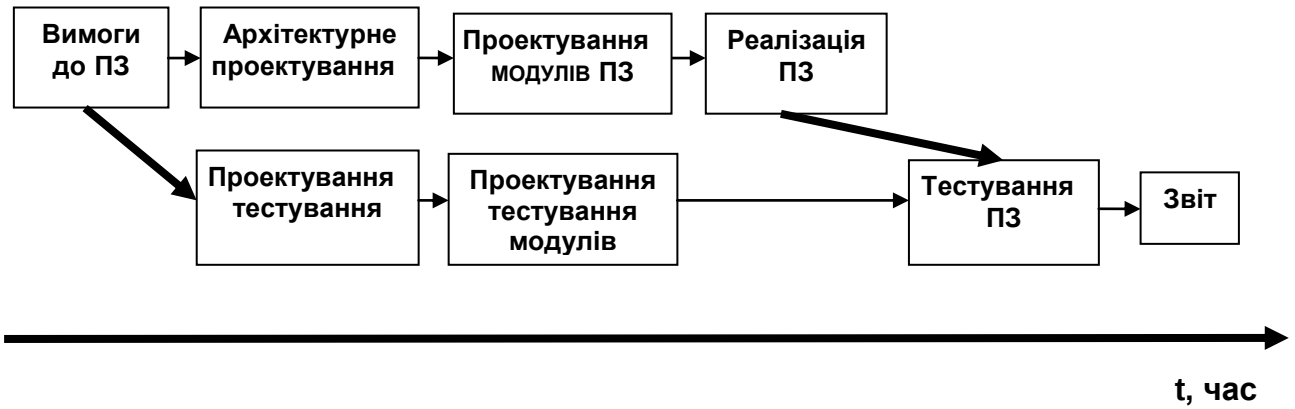

Рисунок 4.2 – Етапи проектування компонентів і тестів

Процес розроблення, в якому проектування тестів так само перенесене «вгору за течією», зазвичай називають розробленням «з пріоритетом тестів».

Переваги такої організації розроблення:

1. Незалежно від того, наскільки ретельно розробляється ПЗ, у ньому будуть помилки. Наприклад, можна випустити з уваги важливу обставину. Можна не помітити і не врахувати будь-які можливі варіанти в силу особливостей складу розуму або образу мислення. Крім того, може просто позначитися втома. Перевірка працездатності програмного забезпечення сторонніми особами

дуже корисна. Ними можуть бути відомі й невідомі Вам обставини, вони можуть помітити щось непомічене раніше. Краще за все, якщо ці люди мислитимуть по-іншому – тоді вони поглянуть на проект з іншого ракурсу, і їм буде простіше побачити те, що Ви не помітили.

2. Досвід показує, що тестування дуже корисне. У першу чергу, воно дозволяє отримати об'єктивні дані. Відклавши проектування тестів до моменту початку тестування, як показано на рисунку 4.1, можна втратити гроші. Помилка в ПЗ не буде знайдена доти, доки будь-який випробувач вже під час тестування не виявить помилку. І тепер для виправлення помилки потрібно буде частково або повністю переписати реалізацію і внести зміни в робочий продукт. Набагато дешевше було б знайти помилку до початку реалізації.
3. Деякі помилки можна виявити ще до створення тестів. Такі помилки зазвичай виявляють реалізатори. Проте і цей варіант не ідеальний. При виявленні помилки реалізацію продукту доведеться призупинити і тимчасово переключитися з обдумування реалізації на обдумування того, яким має бути проект. Це неефективно навіть тоді, коли ролі реалізатора і проектувальника виконує одна і та ж людина, і украй неефективно, коли це різні люди. Розроблення сценарію тестування зручне тим, що допомагає запобігати подібним ситуаціям.
4. Ще одна корисна властивість створення сценарію тестування полягає в тому, що воно сприяє уточненню проекту. Якщо в голові реалізатора виникне питання про те, що ж мав на увазі проектувальник, тест може служити відмінним прикладом очікуваної поведінки програми. У результаті цього в програмному продукті буде менше помилок, спричинених неточним розумінням суті проекту реалізатором.
5. Менше помилок буде навіть у тому випадку, якщо у реалізатора не виникне питань за проектом, а насправді вони можуть виникнути. Наприклад, частенько виникають ситуації, коли проектувальник і реалізатор підсвідомо інтерпретують одну і ту ж річ по-різному. Якщо у реалізатора є не лише проект, але і інструкції відносно поведінки компонента (тесту), то буде набагато вища вірогідність того, що компонент робитиме саме те, що від нього потребують.

Розроблення сценарію тестування ведуть розробники проектів та інші користувачі. Досить часто таке розроблення проводять автори програм. Заслуга цієї системи полягає в тому, що вона дозволяє скоротити обсяг інформації, якою доведеться обмінюватися учасникам проекту.

Розробникові ПЗ і розробникові тестів не треба пояснювати один одному одні і ті ж речі. Більше того, якщо розробленням тестів займається окрема людина, їй доведеться згаяти час на ретельне вивчення проекту, тоді як проект прекрасно відомий розробникові. Нарешті, багато питань типу «Що відбудеться, якщо станеться збій компресора»? виникають як на етапі створення робочих продуктів, так і під час розроблення тестів, тому, можливо, буде ефективніше поставити їх один раз і записати відповіді на папері у вигляді тестів.

Генерація набору даних для тестів дозволить помітити частину помилок, проте, ймовірно, інший співробітник зміг би помітити більше помилок. Серйозність і масштаби цього чинника дуже індивідуальні й сильно залежать від того, наскільки досвідчений розробник.

Ще один недолік того, що ПЗ і тести для нього розробляє одна і та ж людина, полягає у відсутності паралелізму. Хоча розподіл ролей між різними співробітниками приводить до збільшення загального обсягу роботи, в цілому таким чином можна прискорити виконання проекту. Якщо люди бажають скоріше перейти від проектування до реалізації, відкладання цього моменту до розроблення тестів може негативно вплинути на мотивацію колективу. Більше того, у подібних ситуаціях виникає тенденція поверхневого виконання роботи для того, щоб скоріше перейти до наступного етапу.

5 Методи «білого ящика»

5.1 Теоретичний матеріал

Методи «білого ящика» застосовують на етапі кодування, це методи:

- покриття операторів;
- покриття рішень;
- покриття умов;
- покриття рішень і умов

Вони називаються іноді ще методами «скляного ящика» на противагу класичному поняттю «чорного ящика».

При автономному тестуванні методами «скляного ящика» тестувальник (як правило, це програміст-автор) розробляє тести, ґрунтуючись на знанні вихідного коду, до якого він має повний доступ.

5.2 Особливості методів «білого ящика»

Особливості та переваги методів «білого ящика»:

1. **Спрямованість тестування.** Програміст може тестувати програму частинами, передаючи важливі дані. Окремий модуль набагато легше протестувати саме як «скляний ящик».

2. **Повне охоплення коду.** Програміст завжди може визначити, які саме фрагменти коду використані в кожному тесті. Він бачить, які гілки коду ще не протестовані, і може підібрати умови, в яких вони будуть виконані.

3. **Управління потоком.** Програміст завжди знає, яку функцію в модулі слід виконувати наступною і яким має бути її поточний стан. Щоб з'ясувати, чи працює програма так, як він думає, програміст може ввести в неї налагоджувальні команди, які містять інформацію про хід її виконання, або скористатися для цього спеціальним програмним засобом, що називаються відладчиком.

4. **Відстеження цілісності даних.** Програмісту відомо, яка частина програми має змінювати кожен елемент даних. Відстежуючи стан даних (за допомогою того ж відладчика), він може виявити такі помилки, як зміна даних не тими модулями, їх невірна інтерпретація або невдала організація.

5. Внутрішні граничні точки. У вихідному коді видно ті граничні точки програми, які приховані від погляду «ззовні». Наприклад, для виконання конкретної дії може бути використано кілька абсолютно різних алгоритмів і, не заглянувши в код, неможливо визначити, який з них вибрав програміст. Програміст може встановити, яку кількість проміжних даних можна об'єднати, і йому не потрібно буде застосовувати тисячі наборів для тестів.

6. **Тестування,** яке визначається обраним алгоритмом. Для тестування оброблення даних використовують складні обчислювальні алгоритми, крім цього, можуть знадобитися спеціальні технології. Як класичні приклади можна навести перетворення матриці й сортування даних.

Автономне тестування методами «білого ящика» розглядають як частину процесу програмування. Програмісти виконують цю роботу постійно, вони тестують кожен модуль після його написання, а потім ще раз після інтеграції його в систему. Це – тестування, яке визначається обраним алгоритмом.

5.3 Критерії охоплення

Об'єктом тестування може бути не тільки повний шлях виконання програми, а й його окремі ділянки. Для відбору вони користуються спеціальними критеріями, що називаються критеріями охоплення, які визначають цілком реальну велику кількість тестів. Критерії охоплення іноді називають логічними критеріями охоплення, або критеріями повноти. Розглянемо три види критеріїв, які найчастіше використовуються:

- охоплення рядків;
- умов;
- розгалужень.

Обов'язковою вимогою є те, щоб кожен рядок коду був виконаний хоча б один раз.

Ще більш суворим є критерій охоплення умов. За цим критерієм слід перевірити умови по двох напрямах.

Програма повинна проходити не тільки всі рядки коду, але і всі можливі розгалуження. Тестування шляхів програми вважається завершеним, коли обраний критерій охоплення повністю виконаний.

6 Тестування об'єктно-орієнтованого програмного забезпечення

6.1 Метод успадкування

При успадковуванні створюються нові класи, які повторно використовують і розширюють класи, створені раніше. При цьому допускається розширення функцій існуючого коду без реальної модифікації вихідного коду. Ця концепція програмування спричинює деякі проблеми при тестуванні ПЗ.

Похідний клас, або нащадок батьківського класу успадковує можливості, які є у вихідного батьківського класу, хоча при цьому також додаються його власні можливості. Основна ідея полягає у тому, що після написання і налагодження батьківського класу ніхто не захоче змінювати код з побоювання внести нові помилки. Ще одна причина використання успадкування – можлива відсутність доступу до вихідного коду (це відбувається, коли код поширюється у вигляді частини бібліотеки класів).

Розробник може визначити успадкування класів частиною проектування. Точне визначення моменту, коли слід використовувати успадкування, є одним з аспектів гарного об'єктно-орієнтованого проектування. Недоречне застосування успадкування, як і будь-якої іншої техніки написання коду, може ускладнити тестування і налагодження.

На рисунку 6.1 показано приклад структури успадкування для трьох класів:

- A;
- B, нащадку класу A;
- C, нащадку класу B.

Навіть якщо попереднє тестування підтвердило, що батьківський клас сам по собі працює коректно, залишається питання, чи будуть коректно працювати методи, задані в батьківському класі, якщо їх викличе дочірній.

У кожному класі можуть бути використані специфічні методи і залежно від мови програмування успадкування дозволяє дочірньому класу перевизначати (підміняти) деякі з них. Крім того, від методу (загального або приватного) залежить його доступність для дочірнього

класу. Загальні методи доступні для всіх похідних класів, чого не можна сказати про приватні.

Щодо успадкування можна припустити, що методи класу А будуть працювати і тоді, коли вони будуть викликані об'єктом класу В. Таким чином, тестувати потрібно роботу класу А в контексті класу В. Аналогічно потрібно тестувати роботу класу А в контексті класу С і роботу класу В у контексті класу С. Помилки, як правило, виникають при їх взаємодіях у міру переміщення вгору і вниз в ієрархії рівнів класів.

Рисунок 6.1 – Успадкування класів – перевизначення методу

6.2 Метод вирівнювання

При вирівнюванні класів створюється схема, за допомогою якої виділяються всі успадковані функції. На рисунку 6.2 показано всі методи як батьківського, так і похідних класів і вказано видимість для методів кожного класу.

Завдання полягає в тому, щоб протестувати кожен метод у контексті всіх класів, які можуть отримати до них доступ. Використання схеми вирівнювання (рисунок 6.2) допомагає визначити умови тестів (таблиця 6.1). Позначення «Клас – Метод» прийнято для того, щоб зв'язати метод з класом, в якому він задається. У першій

умові тесту сказано, що потрібно протестувати метод м1 класу А в контексті класів А, В і С.

Описана вище методологія дозволяє встановити, які методи доступні для похідних класів.

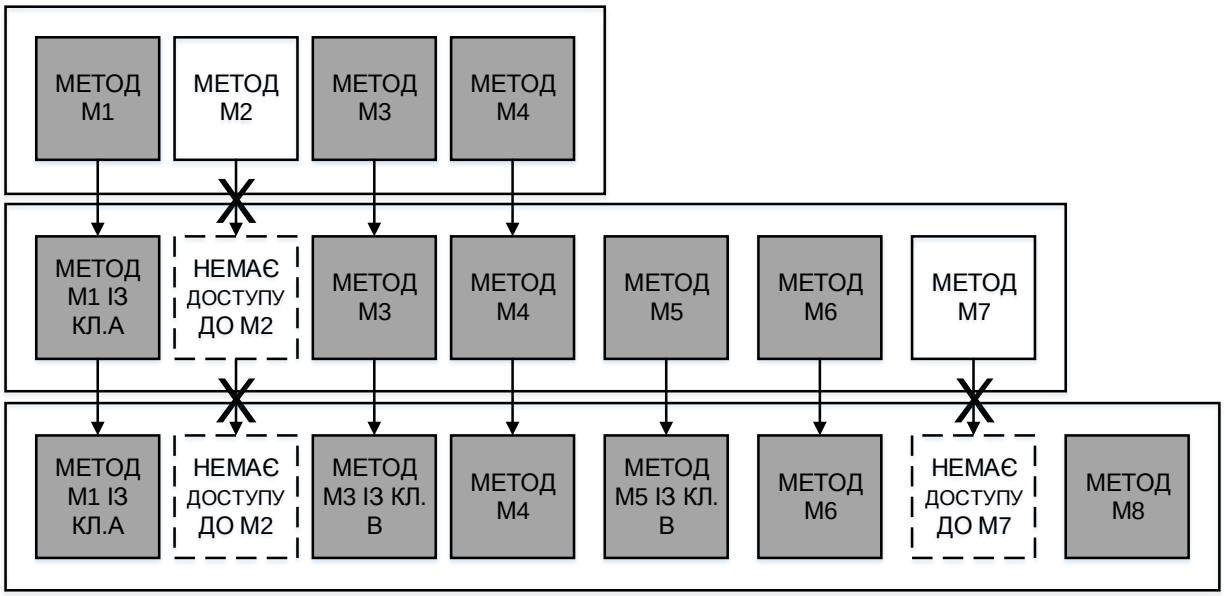

Рисунок 6.2 – Вирівнювання ієрархії класів

Складнощі проведення тестів. Наступний етап завдання тестових прикладів – розроблення середовища, в якому будуть проводитися тести. Для цього може знадобитися спеціальне тестове ПЗ (драйвер), яке буде активізувати тестоване ПЗ. Цей драйвер виконує такі функції:

- упаковку вхідних та інших даних;
- активізацію тестованого ПЗ;
- реєстрацію вихідних результатів.

В об'єктно-орієнтованому проектуванні для приховування складності використовується інкапсуляція і, таким чином, обмежується доступність даних і методів поза класом. Отже, в інших класах неможливо використовувати ці дані й методи без реалізації будь-якого спеціального доступу.

Драйвер повинен мати можливість контролювати і спостерігати за внутрішніми даними і методами класу. Існує кілька способів, за допомогою яких можна цього досягти, а саме:

- правильним вибором методу тестового прикладу для кожного класу;
- створенням паралельних, ідентичних вихідним класів, прикладів, які містять додатковий код, необхідний для тестування;
- створенням дочірніх класів, які успадковують тестовані методи.

Усі ці підходи дозволяють видозмінювати класи і, отже, модифікувати реальний додаток. Важливо, щоб середовище тестування було якомога ближче до середовища розроблення, інакше не можна буде повністю протестувати кінцевий продукт.

Драйвери для тестування класів мають безліч форм, тому перш ніж вибрати схему драйвера, слід проаналізувати різні варіанти.

Створення ефективного драйвера, здатного для повторного використання, потребує серйозного планування і значних зусиль

7 Проектування тестування інтерфейсу «ПЗ-користувач»

7.1 Засоби взаємодій

Розробнику цього інтерфейсу необхідно вирішити дві головні проблеми:

• яким чином користувач буде вводити дані в систему;

• як дані будуть видані користувачеві.

«Правильний» інтерфейс повинен забезпечувати взаємодію з користувачем і подання інформації, тобто введення команд і даних у систему.

Усі види взаємодії можна віднести до одного з п'яти основних стилів взаємодії.

1**. Безпосереднє маніпулювання**. Користувач взаємодіє з об'єктами на екрані. Наприклад, для видалення файла користувач просто перетягує його в кошик.

2. **Вибір з меню.** Користувач вибирає команду зі списку пунктів меню. Дуже часто обрана команда впливає тільки на той об'єкт, який виділений (обраний) на екрані. При такому підході для видалення файла користувач спочатку вибирає файл, а потім – команду на видалення.

3. **Заповнення форм.** Користувач заповнює поля екранної форми. Деякі поля можуть мати своє меню (меню, що випадає, або списки). У формі можуть бути командні кнопки, при натисканні мишею на яких ініціюють деяку дію.

4. **Командна мова.** Користувач вводить конкретну команду з параметрами, щоб вказати системі, що вона має далі робити.

5. **Природна мова.** Користувач вводить команду на природній мові.

Кожен з цих стилів взаємодії має переваги і недоліки і найкращим чином підходить для різних за типами програм і різним категоріям користувачів. В одному додатку може використовуватися одночасно кілька різних стилів.

Нижче перелічені основні переваги та недоліки названих стилів взаємодії і вказані типи додатків, в яких вони зазвичай використовуються.

Безпосереднє маніпулювання. Швидке й інтуїтивно зрозуміле. Складна реалізація, використання у відеоіграх. Цей стиль легкий у вивченні, де є зоровий об'єкт.

Вибір з меню. Скорочення кількості режимів. Повільний варіант для досвідчених користувачів системи. Головним чином, застосовується у сервісних додатках.

Заповнення форм. Уведення з клавіатури клавішами, обраними користувачами. Має мінімальну складність, якщо меню складається з невеликої кількості вкладених пунктів.

Командна мова. Потужна і гнучка, важка у вивченні, операційні системи обов'язково містять командну мову, складно запобігти помилки введення.

Природна мова. Підходить недосвідченим користувачам, потребує процесу інтерпретації, системи зберігання, може бути трудомістким при ручному наборі даних. Використовується в системах зі штучним інтелектом.

7.2 Типи помилок

Типи помилок і коментарі до характеристик їх прояву перелічено у таблиці 7.1.

Таблиця 7.1 – Типи помилок

№ п/п	Тип помилки	Коментарі
1	Функціональна	Важко або неможливо домогтися від ПЗ чогось, що очікує від нього користувач
2	Надлишкова функціональність	Універсальна система, призначена для виконання занадто великої кількості різноманітних завдань, складна і у вивченні, і в експлуатації. Користувачі постійно забувають, як виконати ту чи іншу дію
3	Помилкове враження про набір функцій	Керівництво користувача і маркетингова політика ні в якому разі не повинна створювати у користувача враження, що програма може більше, ніж насправді
4	Неадекватність реалізації базових функцій	Якщо до однієї з ключових функцій ПЗ неможливо звернутися і якщо воно реалізовано занадто вузько або повільно працює
5	Пропущена функція	У програмі відсутня описана в специфікації або очевидна необхідна функція
6	Неправильно працює функція	Функція ПЗ має виконувати одне, а робить щось інше
7	Функція має бути реалізована користувачем	Коли деякі компоненти, що реалізують будь-які можливості, користувач реалізує сам (у цьому випадку це інструментальний набір, а не програма)
8	Програма не робить того, що очікує від неї користувач	Наприклад, необхідно впорядкувати список, а програміст не врахував провідних прогалин або контексту

Продовження таблиця 7.1 – Типи помилок

№ п/п	Тип помилки	Коментарі
9	Пропущена інформація	Усе, про що користувачеві необхідно знати, має відбитися на екрані
10	Відсутність на екрані інструкцій	Наприклад, як визначити назву програми, як вийти з неї, яку клавішу натиснути, щоб отримати довідкову інформацію?
11	У користувача завжди має бути друкована документація.	Недосвідчений користувач не повинен повністю залежати від документації
12	Стан, з якого складно знайти вихід	Якщо користувачеві складно визначити, як вийти з небажаного стану
13	Відсутність курсора	Курсор указує на активну точку екрану, тому він повинен постійно бути присутнім
14	Програма не розпізнає введення	Якщо користувач виробляє введення тексту, символи повинні негайно відображатися на екрані
15	Тривала відсутність активності	Бажано відображати індикатор, який показує, яка частина завдання виконана і скільки часу залишилося
16	Неможливо визначити, чи виконана дана програмі команда	ПЗ може виконати команду не тоді, коли цього очікує користувач, або не з'являться на екрані внесені зміни
17	Неможливо визначити, що один документ відкритий кілька разів	Модуль, що відкриває кілька документів, повинен перевіряти кожен відкритий документ і повідомляти користувачеві, якщо він відкривається повторно
18	Неправильна чи утруднена для користувача інформація	Незначні помилки або недостатньо докладні повідомлення ПЗ, через які користувач може зробити неправильні узагальнення або висновки
19	Невдалі метафори	Наприклад, метафорою є піктограма із зображенням кошика для паперів, яка використовується для видалення файлів. Якщо віддалений файл можна витягнути з кошика, метафора правильна
20	Неточні назви команд або функцій	Якщо в російській або англійській мовах за словом закріпилося певне значення, названа цим словом команда повинна йому відповідати
21	Надлишкова інформація	Велика кількість технічних подробиць і користувач в ній загубився
22	Коли зберігаються дані	Відразу під час введення, або при виході, або за окремою командою, через якийсь час
23	Кілька назв однієї функції	Одна функція не повинна мати в ПЗ кілька визначень
25	У довідковій системі багатослівність	Повідомлення мають бути простими й короткими, нейтральними і коректними. Слова: **аварія, збій**, **втрата даних** украй небажані на екрані

8 Оцінювання якості програм на основі метрик Холстеда

8.1 Теоретичний матеріал

Оцінювання складності програмних продуктів. Інтегральна система оцінювання не тільки складності, але й якості програм у цілому є система метрик, запропонована Холстедом. Способи оцінювання та види складності програмного продукту можна розділити на групи, показані на рисунку 8.1.

Статичні складові складності, характерні для етапів розроблення програмного продукту, і визначають їх трудомісткість.

Рисунок 8.1 – Система метрик Холстеда.

Динамічні складові складності програмного продукту проявляються в процесі його виконання.

Трудомісткість розроблення програмного продукту може оцінюватися двома способами:

- за інтегральними характеристиками складності, що визначаються за зовнішніми параметрами додатків, які не враховує його внутрішню структуру (підхід «чорного ящика»);
- за структурними характеристиками складності, що враховує внутрішню структуру модулів і залежних від складності маршрутів (потоків управління), складності інформаційних зв'язків, потоків даних (підхід «білого ящика»).

Динамічна (або обчислювальна) складність характеризує процес виконання програми і має три взаємозв'язаних складових:

- тимчасову – визначається часом виконання програми або часом його реакції на запит користувача;
- програмну – визначається складом і способом взаємодії процедур або модулів, що утворюють додаток, а також можливістю їх розміщення в кеш-пам'яті, основній пам'яті або на диску;
- інформаційну – визначається складністю організації даних і доступу до них, а також можливістю їх розміщення в кеш-пам'яті, основній пам'яті або на диску.

Інтегральною системою оцінювання не тільки складності, але і якості програм в цілому є система метрик, запропонована Холстедом.

8.2 Вимірні властивості алгоритмів

Виходячи з ідентифікації операторів і операндів, треба визначити вимірні категорії. Вони визначаються метриками, за допомогою яких можуть бути отримані основні характеристики якості модулів.

До складу вимірних властивостей будь-якого уявлення алгоритму (або програми) можуть бути включені такі метричні характеристики:

η_1 – кількість простих (унікальних) операторів, які з'являються в даній реалізації;

η_2 – кількість простих (унікальних) операндів, які з'являються в даній реалізації;

$\mathbf{N_1}$ – загальна кількість усіх операторів, які з'являються в даній реалізації;

$\mathbf{N_2}$ – загальна кількість усіх операндів, які з'являються в даній реалізації;

$\mathbf{f_{1j}}$ – кількість входжень **j**-го оператора, де $\mathbf{j = 1,2,3 \ ... \ \eta_1}$;

$\mathbf{f_{2j}}$ – кількість входжень **j**-го операнда, де $\mathbf{j = 1,2,3 \ ... \ \eta_2}$.

На основі метричних характеристик для модуля треба визначити:

словник η $\quad \eta = \eta_1 + \eta_2$

і довжину **N** $\quad N = N_1 + N_2$ реалізації модуля.

Відповідно до даних визначень повинні виконуватися такі три співвідношення:

$$N_1 = \sum_{j=1}^{j=\eta} f_{1,j},$$
$$N_2 = \sum_{j=1}^{j=\eta} f_{2,j},$$
$$N = \sum_{i=1}^{i=2} \sum_{j=1}^{j=\eta} f_{i,j}.$$

8.3 Довжина модуля

Використовуючи визначені параметри модуля, треба отримати рівняння для оцінки кількісного співвідношення між довжиною програми **N** і словником η.

Рядок довжини **N**, який створений символами, що належать до словника η, має підпорядковуватися низці обмежень.

Вимога, згідно з якою кожен символ словника повинен з'явитися щонайменше хоча б один раз, гарантує виконання умови

$$\eta \leq N,$$

що визначає нижню межу для **N,** виражену через η**.**

Знайдемо верхню межу для **N.** Розіб'ємо рядок довжиною **N** на підрядки довжиною η**.** Розділена таким чином програма для ПК виявляється такою, що складається з **N/**η операторів довжиною η кожен.

Якщо рядок не містить двох однакових підрядків довжиною η, то це – шукана верхня межа.

Отже, якщо загальна підвиразів довжини η потрібно модулю більше одного разу, то присвоювання його окремому операнду збільшить η (кількість типів операторів) на одиницю.

Кількість можливих комбінацій з η елементів, узятих по η за один раз, становить:

$$N \leq \eta^{\eta+1} .$$

Якщо врахувати, що оператори й операнди, як правило, чергуються, то можна отримати інше співвідношення:

$$N \leq \eta \times \eta_1^{\eta 1} \times \eta_2^{\eta 2} .$$

Верхня межа для цієї нерівності має містити не тільки впорядковану множину з **N** елементів, які являють собою досліджувану програму, але і його всілякі підмножини. Сім'я всіляких підмножин з **N** елементів містить 2^N елементів

Відповідно, можна прирівняти кількість можливих комбінацій з операторів і операндів (яка дорівнює кількості подстрок **N/**η**)** кількість підмножин з **N** елементів і висловити довжину реалізації алгоритму через його словник. З рівняння

$$2^N = \eta_1^{\eta 1} \times \eta_2^{\eta 2}$$

отримуємо

$$N = \log_2 (\eta_1^{\eta 1} \times \eta_2^{\eta 2})$$

або

$$N = \log_2 \eta_1^{\eta 1} + \log_2 \eta_2^{\eta 2} .$$

Це дає рівняння оцінки довжини:

$$\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2 . \qquad (8.3.1)$$

У цьому виразі символ **N** забезпечили $\wedge$ для того, щоб відрізняти обчислену (теоретичну) оцінку довжини від значення **N,** отриманого безпосереднім вимірюванням (дослідної довжини).

Вираз (8.3.1) являє собою ідеалізовану апроксимацію виміряної довжини $N=N_1+N_2$, яка справедлива для програм, які не містять стилістичних помилок.

8.4 Обсяг програми

Важливою характеристикою модуля є його розмір, який повинен бути вимірюваною величиною.

Мінімальна довжина залежить тільки від кількості елементів у словнику η**.** У загальному випадку $\log_2 \eta$ є мінімальна довжина (у бітах) усієї програми. Під бітом тут розуміється логічна одиниця інформації - символ, оператор, операнд –- те, чим оперує програміст при створенні програми.

Відповідна метрична характеристика розміру будь-якої реалізації будь-якого алгоритму, що називається об'ємом **V,** може бути визначена як

$$V = N \log_2 \eta, \qquad (8.3.2)$$

де $N = N_1 + N_2$ – довжина її реалізації, а $\eta = \eta_1 + \eta_2$ –- її словник.

8.5 Потенційний обсяг

Позначивши відповідні програмні параметри можливо найкоротшою або найбільш стислою формою алгоритму зірочками, з рівняння (8.3.1) отримаємо, що мінімальний (або потенційний) обсяг **V**

$$V^* = (N_1^* + N_2^*) \log_2(\eta_1^* + \eta_2^*) . \qquad (8.3.3)$$

Але в мінімальної формі ані оператори, ані операнди не потребують повторень, тому

$$N_1^* = \eta_1^* ; \quad N_2^* = \eta_2^* ,$$

що дає

$$V^* = (\eta_1^* + \eta_2^*) \log_2(\eta_1^* + \eta_2^*) .$$

Крім того, відома мінімально можлива кількість операторів η_1^* для будь-якого алгоритму. Кожен алгоритм має містити один оператор для імені функції або процедури і один як символ присвоювання або угруповання, тобто мінімальне значення $\eta_1^*=2$**.**

Рівняння тепер набуде вигляду

$$V^* = (2 + \eta_2^*) \log_2(2 + \eta_2^*) , \qquad (8.3.4)$$

де η_2^* має являти собою кількість різних вхідних і вихідних параметрів.

З рівняння (8.3.4) випливає, що потенційний обсяг V^* будь-якого алгоритму не повинен залежати від мови, на якій він може бути реалізований.

Якщо η_2^* розцінюється як кількість однакових за змістом (ненадлишкових) операндів, то V^* виявляється найбільш корисним заходом утримання алгоритму.

9 Визначення інтелектуального змісту програми

9.1 Рівень програми

Поняття рівня програми відомо відтоді, як перші «мови високого рівня» отримали свою назву, але воно має бути виражене кількісно, або приведено до вимірного значення.

Рівень мови і рівень програми є різними поняттями, хоча і тісно пов'язаними.

Виведені в попередній лабораторній роботі вирази для визначення обсягу **V** програми і потенційного обсягу **V *** підказують простий метод кількісного вираження даного поняття.

Якщо записати рівень **L** програми, який є реалізацією будь-якого алгоритму, як

$$L = \frac{V^*}{V}, \qquad (9.1.1)$$

то лише найбільш стислий вираз, який тільки можливий для алгоритму, матиме рівень, що дорівнює одиниці.

Більш об'ємні реалізації матимуть менші значення рівня, так що $L \le 1$.

Перебудувавши вираз (9.1.1), отримаємо

$$V^* = L \times V . \qquad (9.1.2)$$

Тобто при збільшенні обсягу рівень програми зменшується, і навпаки. Однак рівень програми відіграє подвійну роль у визначенні легкості або труднощів її розуміння.

Спеціаліст, якому відомі всі використовувані терміни, зможе вловити ідею тим швидше, чим вище рівень її подання.

9.2 Висновок рівняння рівня програми

Треба визначити рівень програми безпосередньо з її реалізації, не знаючи її потенційного обсягу V^* і не посилаючись на можливе звернення до неї у вигляді виклику процедури.

Розглянувши окремо вплив операторів і операндів на рівень програми, допустимо, що чим більше кількість унікальних операторів, використовуваних у реалізації, то тим нижче її рівень.

Але найменш можлива кількість унікальних операторів дорівнює двом і ці два оператори суть символу функції та оператора присвоювання або угрупованні, тобто у результаті цього отримуємо таке співвідношення операторів:

$$L \sim \frac{\eta_1^*}{\eta_1} . \qquad (9.1.3)$$

Будь-яке повторення імені операнда є причиною на переклад на більш низький рівень реалізації.

Цей ефект можна виміряти, взявши відношення кількісті простих операндів до загальної кількості операндів:

$$L \sim \frac{\eta_1}{N}. \qquad (9.1.4)$$

Об'єднуючи рівняння (9.1.3) і (9.1.4), отримаємо коефіцієнт пропорційності, який є константою і повинен дорівнювати одиниці, оскільки в разі потенційної мови $\eta_1=\eta_1^*$, $\eta_2=N_2$ і **L=1.**

Тоді рівняння рівня програми матиме вигляд

$$\hat{L} = \frac{\eta_1^*}{\eta_1}\frac{\eta_2}{N_2}, \qquad (9.1.5)$$

у якому символ ^ вказує на те, що рівень, який визначається цією формулою, служить апроксимацією рівняння (9.1.1).

9.3 Визначення інтелектуального змісту програм

Розглянуті співвідношення для обсягу й рівня програми приводять до такого висновку. Якщо добуток обсягу на рівень будь-якого алгоритму дійсно не змінюється при вираженні його на різних мовах, то цю властивість можна вважати фундаментальною. Міру того скільки було сказано в програмі, логічніше було б назвати інформаційним змістом програми, але теорія інформації Шеннона раніше інших заволоділа змістом цього терміна, правда, для вказівки лише обсягу повідомлення без урахування його рівня.

Тому «інформаційний зміст» програми замінено поняттям інтелектуального змісту і визначимо які:

$$I = \hat{L} \times V . \qquad (9.1.6)$$

За допомогою рівнянь (9.1.1) і (9.1.5) цей вираз можна подати у вигляді

$$I = \frac{2}{\eta_1}\frac{\eta_2}{N_2} \times (N_1 + N_2)\log_2(\eta_1 + \eta_2) \qquad (9.1.7)$$

де всі члени в правій частині доступні безпосередньому вимірюванню, виходячи з будь-якого виразу алгоритму.

Інтелектуальний зміст, обчислений за допомогою рівняння (9.1.6), приблизно дорівнює потенційному обсягу V^*, отриманому з рівняння (9.1.4).

Отже, оскільки V^* не залежить від мови, на якій виражений алгоритм, то інтелектуальний зміст також не повинен від нього залежати.

Доки параметри I і V^* використовуються тільки як допоміжні величини, вони є дуже корисними метричними характеристиками, які задають міру того, «скільки» було визначено, але нічого не кажуть про важливість визначеного.

Тобто програмні параметри показують, наскільки ефективно написана програма, але вони не визначають, чи потрібно було взагалі її писати.

10 Верифікація і тестування програмних додатків для мобільних пристроїв

10.1 Етапи роботи.

1. Аналіз предметної області. Навести результати аналізу.
2. Формування функціональних вимог до додатка. Навести структуровані й сформовані вимоги до додатка.
3. Розроблення для користувача інтерфейсу, призначення якого– швидке і точне введення інформації. Виконувати розроблення треба, аналізуючи:
   - розміри екранного простору;
   - розмір пам'яті;
   - витрати процесу створення програми.

Точність уведення та високу продуктивність уведення підвищить своєчасний аналіз таких факторів:

- використання однієї або двох рук;
- кількість і розмір кнопок;
- значення «за замовчуванням»;
- комбінацію клавіш, вкладки (для переходу між набором списків).

Особливу увагу треба приділити аналізу моделі навігації (рисунок 10.1). Аналіз її особливостей дозволить значно поліпшити ефективність тестування. Необхідно розробляти тести, що розраховуються на певний рівень моделі.

Особливості призначеного для користувача інтерфейсу, які впливають на модель навігації:

- детальне уявлення, усі «абзаци» показані на екрані, тому окремі пункти повинні завантажуватись у прямому і зворотному напрямках;

• загальне уявлення, на екрані виведений один список усіх «абзаців» і можна вибрати один елемент списку; не допускається одночасне використання детального та загального уяавлення;

• прямолінійне уявлення – після натискання клавіш переходу:

- зліва-направо,
- зверху-вниз,
- клавіша «**ТАК**» вперед,
- клавіша «**НІ**» назад;

• елементи керування з урахуванням їх розмірів (дрібні чи ні);

• спливаюча клавіатура для введення даних.

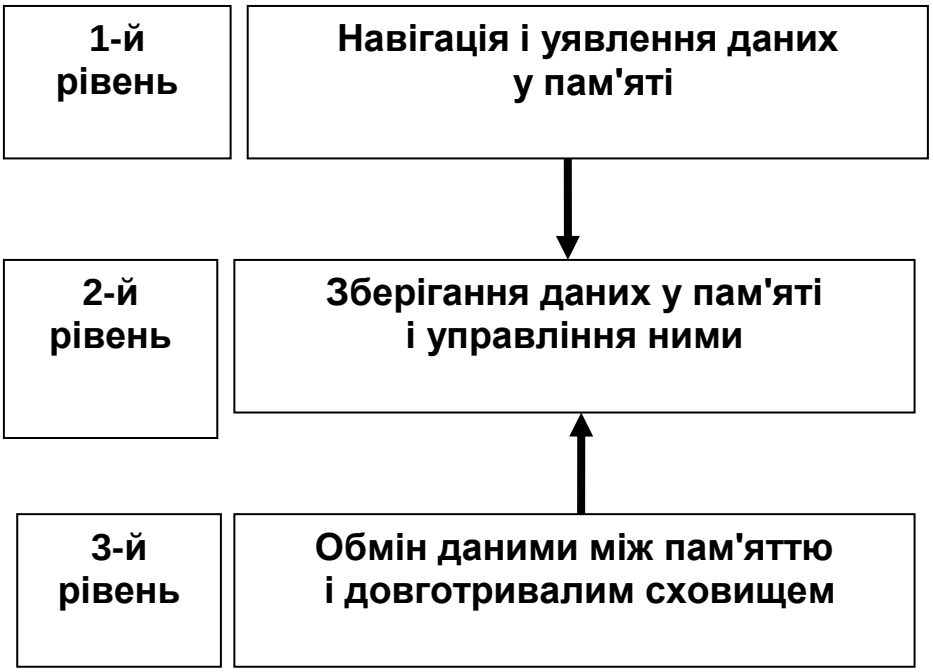

Рисунок 10.1 – Модель навігації

4. Розроблення моделі даних.

Існують дві моделі:

• універсальна абстрактна модель;

• призначена для користувача модель для роботи з даними користувача, тобто управління конкретними розробленими функціями і діями.

Способи доступу до даних:

• додаток - сховище;

• управління даними, ресурсами (база даних, сховище, файли).

Два способи роботи з даними:

• запит до БД:

• обмін даними, які знаходяться у пам'яті.

Наповнення БД:

• динамічний спосіб запису даних під час експлуатації додатків;

• читання зі створених раніше файлів;

• копіювання даних у пам'ять.

Доступ до довготривалих даних через WEB-служби.

Для полегшення роботи з Web-службами передбачені бібліотеки різних рівнів.

У порядку підвищення рівня абстракції:

- Можливість виконувати HTTP / HTTPS-запити.
- Можливість генерувати і аналізувати XML-документи.
- Можливість генерувати і аналізувати повідомлення SOAP.
- Можливість автоматично генерувати проксі-код для додатків клієнтів Web-служб.

5. Розгортання мобільного додатка.

Для підвищення надійності функціонування програми тестування обов'язково треба виконувати:

- у реальних умовах;
- тільки на фізичних пристроях.

10.2 Правила тестування

1. Правила тестування комунікаційних процедур.

А. Тестувати й налагоджувати комунікаційний код простіше, якщо він виконується в синхронному режимі потоком інтерфейсу користувача. Тому доцільніше виявити всі помилки в коді, виконуючи його синхронно з призначеним інтерфейсом для користувача, а потім приступити до тестування коду при його виконанні фоновим потоком.

Б. Необхідно ускладнювати умови тестування комунікаційних процедур, викликаючи їх в асинхронному режимі в критичних ситуаціях.

Код, що виконується фоновим потоком, завжди складніший коду, що викликається синхронно. Кращий спосіб створення дійсно надійних систем - це тестування коду в спрощеному оточенні, призначеному для переведення коду в спеціально ускладнені (критичні) стани і моніторинг виконання коду для виявлення незвичайних ситуацій.

В. Після тестування коду необхідно перетворити комунікаційні процедури, щоб вони могли виконуватися в робочому фоновому потоці додатків.

Виконавши тестування базового комунікаційного коду в синхронному режимі, потім його тестування в асинхронному режимі, він буде готовий до переведення в асинхронну операцію, виконувану в додатку, що розробляється.

2. Тестування відмовостійкості додатка.

Перш за все треба визначити всілякі умови виникнення помилок під час передачі даних, спеціально порушивши всі види зв'язків:

- збої зв'язку за допомогою коду на стороні клієнта;
- збої зв'язку за допомогою коду на стороні сервера.

Додаток, незважаючи на збої, має надати кінцевому користувачеві інтерфейс, який зберігає здатність до відгуку та можливість відновлення роботи програми.

3. Обов'язково треба проаналізувати відновлення умов доступу до мережі після усунення додатком наслідків збою.

Перелік умов і даних, які обов'язково мають бути вказані.

Параметри пристрою.

Мікропроцесор, накопичувачі.

дисплей:

- ємнісний;
- резистивний.

Обсяг оперативної пам'яті .

Засоби зв'язку:

- мобільний internet;
- wi-fi;
- інший бездротовий зв'язок.

Версія OS Android.

Способи розроблення.

Указати способи розроблення:

- модель серверного WEB-додатка;
- модель інтелектуального клієнта:
    - управління введенням інформації і її обміном;
    - додаткові функціонали клієнта.

Указати розташування даних:

- оперативна пам'ять;
- внутрішня пам'ять;
- CD-карта.

Визначення та тестування положення пристрою

При запуску програми перевірити стан, в якому знаходиться пристрій.

Варіанти визначення такі.

- Неможливо визначити положення.
- Пристрій в режимі з кнопкою «Вихід» внизу.
- Пристрій в режимі з кнопкою «Вихід» вгорі.
- Пристрій в горизонтальному положенні, кнопка «Вихід» праворуч.
- Пристрій в горизонтальному положенні, кнопка «Вихід» зліва.
- Пристрій паралельно землі, екраном вгору.
- Пристрій паралельно землі, екраном вниз.
- Надання пристрою бажаного положення.

Контроль використання елементів управління.

Записати призначення кожного елемента управління, призначення кожної кнопки, оскільки кожен елемент управління – це унікальний конкретний елемент.

У додатку визначити їх ієрархію для здійснення дій над ними:

- у прямому напрямку;
- у зворотному напрямку;
- комбінації клавіш.

Тестування мультизадачності.

Для імітації виходу користувача з програми за допомогою кнопки «Вихід» треба передбачити метод, який згортає програму і викликає її знову через певний період часу.

1. Запуск.
- Натискання на кнопку «Старт».
- Натискання на кнопку «Пауза»:
    - збереження екрана;
    - резервне копіювання на зовнішньому носії;
    - запис проміжних даних у файл.
- Натискання на кнопку «Вихід».
- Натискання на випадкову кнопку «Меню».
- Вибір фону заставки.
- Натискання на кнопку повторного запуску після «Паузи»:
    - відновлення екрана;
    - читання інформації з зовнішнього носія;
    - читання проміжних даних з файла.

2. Вхід у конкретне «Меню».

2.1. Вибір випадкової кнопки.
- Натискання – легке.
- Натискання – сильне.

2.2. Перетягування бігунка на конкретну позицію.
- Натискання – легке.
- Натискання – сильне.
- Натискання на елемент управління «Обертання»:
    - за годинниковою стрілкою;
    - проти годинникової стрілки.
- Натискання на елемент «Зміна» напряму руху:
    - зліва направо;
    - справа наліво;
    - по діагоналі (вгору-вниз).

2.3 Натискання на елемент управління «Звук»:
- збільшення;
- зменшення.

2.4 Масштабування розміру кнопок:
- збільшення;
- зменшення.

Тестування комунікаційних процедур.

Визначити і записати всі умови (набори даних для тестів) тестування комунікаційних процедур, викликаючи їх в синхронному режимі в критичних ситуаціях використання тестів.

Виконати тестування базового комунікаційного коду в синхронному режимі, потім – його тестування в асинхронному режимі,

Указати умови відновлення доступу до мережі після усунення додатком наслідків збою.

Тестування виклику Web-служб з мобільного пристрою.

Для полегшення роботи з Web-службами передбачені програмні бібліотеки різних рівнів.

У порядку підвищення рівня абстракції:

- Можливість виконувати HTTP / HTTPS-запити.
- Можливість генерувати й аналізувати XML-документи.
- Можливість генерувати й аналізувати повідомлення SOAP.
- Можливість автоматично генерувати проксі-код для додатків клієнтів Web-служб.

Тестування умов і обсягу зберігання даних.

1. Указати розташування і конкретний обсяг даних:
   - файл;
   - оперативна пам'ять;
   - внутрішня пам'ять;
   - CD-карта.
2. Навести всі набори даних для всіх тестів.

Указати рівень взаємодії додатків із зовнішніми ресурсами.

Перш за все обґрунтувати вибір з наведеного списку рівень взаємодії додатків із зовнішніми ресурсами:

1) замкнуті обчислювальні системи;

2) кооперативні обчислення спільно з операційною системою;

3) кооперативні обчислення спільно з іншими додатками, що виконуються на тому ж пристрої;

4) кооперативні обчислення з інтенсивним використанням мережі.

Тестування енергоспоживання.

Указати:

- розмір зарядки батареї;
- рівень освітленості:
  - у приміщенні:
    - світле;
    - темне;
  - на вулиці в сонячний день;
- час (добове):
  - вдень;
  - пізно ввечері;

- способи зв’язку;
- кількість сот (рівень сигналів Wi-Fi);
- стан пристрою:
    - статичний (пристрій не рухається);
    - динамічний (переміщення в машині, поїзді, «на ходу» тощо).

ВИСНОВКИ. У результаті тестування користувачам надається інформація (протоколи, акти) про стан інформаційного обміну даними між активними учасниками цього процесу.

Застосування засобів шифрування і дешифрування при передачі даних потребує виконання додаткових обчислень на обох кінцях лінії, що буде впливати на продуктивність. Якщо додаток потребує безпечної передачі даних, то проектування і тестування відповідних засобів безпеки слід починати якомога раніше.

11 Тестування WEB-додатка

11.1 Особливості тестування WEB-додатків

Проблеми тестування.

1. Постійно змінюються технології.
2. Короткий цикл випуску.
3. Велика кількість користувачів при запуску WEB-вузла.
4. Доступність WEB-вузла протягом 24 годин.
5. Неможливість контролю призначеного для користувача середовища запуску.

Проблеми складності тестування WEB-вузла.

- Одночасне оброблення кількості замовників.
- Безпека інформації про покупців.
- Збереження БД.
- Інформація про товари.

11.2 Плани тестування WEB-додатків

11.2.1 План виконання системного тестування

Програмне забезпечення буде складатися з трьох об'єктів тестування:

- клієнтська частина;
- серверна частина програми;
- серверна частина бази даних.

Мета проведення системного тестування клієнтської частини:

- тестування інтерфейсу «ПК - користувач»;
- перевірка роботи всіх режимів і функцій.

Мета проведення системного тестування серверної частини:

- перевірка сумісності системи;
- перевірка оброблення запитів клієнтської частини програми;

- перевірка виконання з'єднання з базою даних;
- перевірка правильності виконання запитів до бази даних.

Мета проведення системного тестування серверної частини бази даних:

- перевірка створення таблиць;
- перевірка роботи вставки даних у таблиці;
- перевірка зміни даних у таблицях;
- перевірка виконання запитів;
- перевірка роботи тригерів.

Критеріями завершення тестування є такі:

- кількість створених таблиць;
- кількість записів у кожній таблиці;
- відповідь на запит клієнта протягом 10 мс;
- кількість екранних форм.

Засоби тестування: Eclipse 3.3, MySQL 5.0, браузери FireFox, Internet Explorer.

Конфігурація апаратури для проведення тестування серверної частини програми і серверної частини баз даних:

- процесор типу Pentium III і вище;
- оперативна пам'ять від 2048 Mb;
- НЖМД місткістю від 40 Gb.

Процедури відстеження тестування. Основним засобом тестування серверної частини БД є СУБД MySQL 5.0. Відстеження серверної частини програми буде здійснюватися за допомогою середовища Eclipse 3.3, відстеження тестування клієнтської частини буде здійснюватися за допомогою браузерів FireFox, Internet Explorer.

11.2.2 План виконання інтеграційного тестування

Об'єкт тестування: модулі системи і взаємозв'язок між ними.

Мета інтеграційного тестування:

- перевірка інформаційних зав’язків модулів;
- перевірка наявності таблиць і зав’язків між ними;
- знаходження помилок в інтерфейсі між модулями;
- знаходження помилок в інтерфейсах між функціями і модулями.

Критерій закінчення – відсутність помилок в інформаційних зв'язках при проходженні висхідного тестування, внесення запису в базу даних, послідовна видача інформації.

Спочатку тестуються модулі занесення, зміни і видалення інформації, яка використовується в ПЗ, потім – модулі оброблення інформації, модуль перевірки доступу користувача і модуль входу в систему.

Засоби тестування: Eclipse 3.3.

Вказуються модулі, які потенційно містять максимальну кількість помилок.

Конфігурація апаратури повинна бути вказана.

Після виправлення помилок передбачається регресійне тестування з метою виключення нових помилок.

11.3 Виконання тестування WEB-додатка

Основні категорії тестування WEB-вузлів.

1. Функціональні можливості.
2. Практичність.
3. Навігація.
4. Форма.
5. Вміст сторінки.

11.3.1 Тестування функціональних можливостей

Суть. Тестування функцій взаємодії з користувачем. Це контроль.

- Форми.
- Пошуку.
- Спливаючих (тимчасових) робочих вікон.
- Обробки замовлення покупця.
- Оплата в інтерактивному режимі.

Треба звернути увагу на такі особливості.

1. Реальні вхідні дані відрізняються (їх набагато більше) від загального опису даних. Тобто для одного тесту існує декілька наборів вхідних даних.

2. Вміст сторінок постійно змінюється.

3. Запити до сторінок – динамічна величина.

11.3.2 Тестування практичності

Суть. Тестування реальних дій користувача. Основні етапи тестування практичності такі.

1. Визначення завдань WEB-вузла, які впливають на вибір основних запитів і розроблення сценарію для перевірки роботи.

Приклад потреб користувача:

- отримати конкретну інформацію;
- купити товар;
- дізнатися про стан рахунку в банку;
- грати на біржі.

2. Визначення групи користувачів, для яких призначений WEB-вузол, тобто встановлення конкретних характеристик, параметрів, категорій користувачів.

3. Завдання тестів для виконання тестування практичності.

ВИСНОВОК. Виконання тестування практичності залежить від предметної області та особливостей використання WEB-додатків.

11.3.3 Тестування навігації

Суть. Це тестування посилань, перевірка заголовків «над курсором мишки». Основні проблеми тестування навігації:

1. Перехід на сторінку і з неї.
2. Прокрутка сторінок.
3. Клацання на розгорнутих і згорнутих зображеннях для контролю того, що вони працюють.
4. Тестування всіх посилань (як всередині, так і зовні WEB-вузла), щоб перевірити їх коректність і переконатися, що немає розірваних посилань.
5. Контроль того, що вікна з великою кількість блоків обробляються так, як якщо б кожна з них містила тільки один блок.
6. Підтвердження сумісності та узгодженого використання клавіш, клавіш і швидкої дії миші.

ВИСНОВОК. Необхідно тестувати всі посилання WEB-вузла, включаючи текстові та графічні гіперпосилання.

11.3.4 Тестування форм

Тестування форм відбувається з таких дій.

- Використання клавіш табулятора для перевірки того, що форма проходить по полях у прямому і в зворотному напрямках.
- Тестування граничних значень.
- Перевірка правильного поновлення формами інформації.
- Перевірка коректного пошуку формами помилкових даних.

Приклади призначення тестів.

- Перехід табулятора від поля до поля.
- Уведення максимальної кількості символів, яку можна ввести в поле.
- Перевищення кількості символів, яку можна ввести в даному полі.
- Пропуск інформації в полі.

ВИСНОВОК. Застосування методів «чорного ящика» – розбиття на класи еквівалентності і аналіз граничних значень – забезпечують ефективне визначення наборів вхідних даних тестів для перевірки роботи полів

11.3.5 Тестування вмісту сторінки

Суть. Перевірка коректності змісту сторінки з точки зору користувача. Ці тести діляться на дві категорії.

Завдання **першої категорії**: підтвердження правильного функціонування кожної компоненти:

1. Зміст сторінок подано відповідно до вимог.
2. Структура сторінок залишається постійною в різних браузерах.
3. Усі зображення правильно відображаються в різних браузерах.
4. Критичні сторінки зберігають зміст від версії до версії.

5. Посилання змісту всередині і зовні вузла точні.

6. Об'єкти з текстом підказки коректні.

7. На WEB-сторінках витримана палітра кольорів, що є доброзичливою для сприйняття користувачами.

Завдання **другої категорії** – це перевірка змісту на правильність орфографії, граматики, точності визначень і використання термінології, прийнятої в предметній області.

Приклади призначення тестів:

- переглянути зображення, графіку, таблицю;
- помістити курсор біля кожного об'єкта;
- виконати пошук за категоріями.

Крім розглянутих п'яти основних категорій тестів для WEB-вузлів, необхідно виконати такі види тестування.

11.3.6. Тестування конфігурації і сумісності

Приклади призначення тестів:

- Установка декількох сеансів з WEB-вузлом.
- Створення скороченої сторінки і спроба доступу до неї.

11.3.7 Тестування надійності і доступності

Приклади призначення тестів:

- Тестування продукції при використанні її в години пік і не в години пік.
- Тестування під час модернізації БД (декілька користувачів одночасно купують одну і ту ж книгу).

11.3.8. Тестування характеристик

Приклади характеристик, які можуть вплинути на експлуатацію WEB-додатків, тому необхідно їх врахувати при тестуванні:

- Час доби, завантаження, реакції користувача.
- Швидкість доступу в Internet.
- Брак ресурсів через велику кількість користувачів.

11.3.9. Тестування масштабованості

На масштабованість впливають такі типи сценаріїв.

- Наскільки близько середовище тестування збігається з середовищем експлуатації.
- Кількість і тип користувачів під час запуску.

Приклади призначення тестів.

- Тестування найбільш часто використовуваних типів сценаріїв.
- Тестування рівнів користувачів.
- Тестування простої навігації по WEB-вузлу та базові операції з WEB-вузлом.

11.3.10. Тестування завантаженості

Мета. Моделювання відчуття реального світу шляхом генерації безлічі користувачів, які отримали доступ до WEB-сайту.

11.3.11. Тестування тиску

Мета. Підтвердження надійної роботи програми в момент, коли доступна пам'ять і ресурси процесорів близькі до граничних значень.

Приклади призначення тестів.

- Кількість запитів, транзакцій і кбіт в секунду.
- Поточна кількість з'єднань.
- Час на оброблення запиту і формування результату.

11.3.12. Тестування безпеки

Приклади призначення тестів.

- Контроль за правами доступу.
- Контроль захисту інформації від несанкціонованого доступу, які методи використовуються для шифрування інформації.
- Контроль роботи з кредитними картками.

11.3.13. Тестування наскрізних транзакцій

Тести цієї категорії доповнюють тестування практичності, тобто більш докладне послідовне виконання операцій і дій користувача.

11.3.14. Тестування баз даних

Ключові проблеми тестування баз даних:

- цілісність даних;
- достовірність даних, правильний вибір типів даних, відсутність додаткових перетворень;
- оброблення даних:
    - тестування на двох рівнях:
    - адміністративних;
    - призначених для користувача функцій.

Додаток А

Лабораторна робота № 1. План системного тестування

Мета. 1. Навчитися обґрунтовувати вибір категорій проведення системного тестування програмного забезпечення.

2. Навчитися складати план системного тестування ПЗ, яке треба проводити в середовищі розроблення та в середовищі замовника.

3. Спроектувати тестування для контролю всіх вимог, які сформульовані в документі вимог до ПЗ.

1 Хід роботи

1. Вивчити теоретичний матеріал (розділ № 1).
2. Записати обгрунтовано вибрані категорії з цього розділу, які треба перевірити під час системного тестування.
3. Розробити план системного тестування, що містить перераліченні розділи.
4. Оформити звіт, що містить план виконання системного тестування та висновки про виконану роботу.

3 Контрольні запитання

1. Які розділи плану системного тестування?
2. Скільки категорій контролюється при системному тестуванні?
3. Хто бере участь у виконанні системного тестування?
4. Що контролюється при виконанні системного тестування?
5. Чи допустимі негативні результати тестів?
6. Чи існують загальноприйняті методології проектування системних тестів?
7. Чи треба застосовувати всі п'ятнадцять категорій для системного тестування?
8. Яка документація тестується під час системного тестування?
9. Які існують способи виконання тестування ПЗ системи?
10. Що таке тестування цілісності?

Лабораторна робота № 2. Інтеграційне тестування

Мета. Навчитися планувати тестування з метою контролю сполучення між модулями, компонентами, підсистемами, коли основні компоненти зібрані в систему**.**

1 Хід роботи

1. Вивчити теоретичний матеріал (розділ № 2).
2. Вибрати метод виконання тестування.
3. Вибрати методики виконання тестування.
4. Визначити стратегію реалізації тестування.

5. Оформити звіт, що містить обґрунтування вибору методу, способу, стратегії виконання тестування і висновки про виконану роботу.

2 Контрольні запитання

1. Яке призначення інтеграційного тестування?
2. Які завдання інтеграційного тестування?
3. Які методи виконання інтеграційного тестування?
4. Які стратегії виконання інтеграційного тестування?
5. Характеристика висхідної стратегії реалізації тестування.
6. Характеристика спадної стратегії реалізації тестування.
7. Характеристика перевірки взаємодії модулів при монолітному тестуванні.
8. Характеристика перевірки взаємодії модулів при покроковому тестуванні.

Лабораторна робота № 3. Тестування методами «чорного ящика»

Мета. Вивчити тестування програм методами «чорного ящика».

Побудувати функціональні діаграми для двох-трьох функцій або режимів для ПЗ свого варіанту, попередньо склавши необхідний набір ситуацій і ефектів, які визначаються критеріями тестування.

1 Хід роботи

1. Прочитати розділ № 3.
2. Самостійно скласти специфікації (вхідні-вихідні дані із зазначенням типу і діапазону значень) для двох-трьох функцій або режимів для ПЗ свого варіанту.
3. Проаналізувати специфікації, вибравши всі явні і неявні ситуації умов на вході.
4. Скласти список ефектів.
5. Довільно пронумерувати ситуації і ефекти.
6. Побудувати функціональні діаграми.
7. Перетворити функціональні діаграми в таблицю рішень (матриця трасування може бути зворотною).
8. Проаналізувати таблицю рішень щодо неоднозначності.
9. Скласти звіт відповідно до порядку виконання роботи. Зробити висновки.

2 Контрольні запитання

1. Які існують методи «чорного ящика»?
2. Що таке розбиття множини вхідних даних на класи еквівалентності?
3. У яких випадках доцільно застосовувати метод граничних значень?

4. Які переваги дозволяють отримати методи «чорного ящика» для тестування програмних продуктів?

5. Метод побудови «функціональних діаграм» належить до методів «чорного ящика» чи ні?

6. Чи потрібні тестувальнику тексти вихідних кодів для тестування методами «чорного ящика»?

7. Що таке функціональна діаграма?

8. Які базові символи використовуються для запису функціональних діаграм?

9. Що являє собою процедура синтезу таблиці рішень?

10. Що таке процедура побудови функціональної діаграми?

11. Які бувають обмеження на ситуації?

12. Які булеві функції можна застосовувати при побудові функціональних діаграм?

Лабораторна робота № 4. Генерація масиву даних для тесту

Мета. Набути практичного досвіду в оптимізації формування набору даних для виконання одного тесту.

1 Хід роботи

1. Вибрати завдання згідно зі своїм варіантом (номером за списком групи) з таблиці 1.1.

2. Уважно прочитати умови роботи програм. Виявити обмеження на максимальні та мінімальні значення змінних, типи даних, час роботи програми, використану пам'ять.

3. Обрати мову та середовище розроблення програми генератора набору даних для одного тесту.

4. Записати обмеження на вхідні та вихідні параметри (згідно з обраною мовою та середовищем) у вигляді таблиці з полями: параметр, тип, мінімальне, максимальне, реальне, критичне значення.

5. Написати процедури (функції або методи), які генерують набори тестових даних і вихідні значення (рішення) для задачі свого варіанта.

6. За допомогою створеної програмної реалізації генератора тестів для задачі свого варіанту створити комплекти тестів. До кожного комплекту входять два файли: testN.dat (вхідних даних) та testN.ans (вихідних даних).

7. Показати результат роботи викладачу.

8. Оформити документ звіту до лабораторної роботи згідно з правилами кафедри, зміст якого наведено.

1. Постановка завдання для свого варіанту, включаючи формулювання завдання і обмеження на рішення, вхідні і вихідні дані.

2. Вхідні та вихідні дані для завдання із зазначенням типу даних і діапазону можливих значень; опис вихідних значень для «неможливих» вхідних даних (для яких задача не має рішення), якщо такі передбачені у формулюванні завдання.

3. Перелік тестових випадків, які вважаєте необхідними для даної задачі.

4. Опис програмної реалізації генератора наборів даних для тестів для завдання свого варіанту, що містить опис методів, процедур, їх параметрів і значень (якщо такі є).

5. Лістинг програми генератора тестів.

6. Тести, отримані в результаті роботи програми генератора тестів у вигляді пар файлів:

test1.dat test1.ans
test2.dat test2.ans
. . .
testN.dat testN.ans

Таблиця1.1 – Варіанти задач, розміщені на сайті Codeforces

Номер варіанта	Задача
1	https://codeforces.com/problemset/problem/1/B
2	https://codeforces.com/problemset/problem/1/C
3	https://codeforces.com/problemset/problem/2/B
4	https://codeforces.com/problemset/problem/4/A
5	https://codeforces.com/problemset/problem/6/A
6	https://codeforces.com/problemset/problem/7/C
7	https://codeforces.com/problemset/problem/8/B
8	https://codeforces.com/problemset/problem/9/A
9	https://codeforces.com/problemset/problem/10/B
10	https://codeforces.com/problemset/problem/10/C
11	https://codeforces.com/problemset/problem/11/B
12	https://codeforces.com/problemset/problem/13/A
13	https://codeforces.com/problemset/problem/15/B
14	https://codeforces.com/problemset/problem/15/E
15	https://codeforces.com/problemset/problem/16/C
16	https://codeforces.com/problemset/problem/17/A
17	https://codeforces.com/problemset/problem/18/B
18	https://codeforces.com/problemset/problem/19/B
19	https://codeforces.com/problemset/problem/20/B
20	https://codeforces.com/problemset/problem/23/A
21	https://codeforces.com/problemset/problem/27/A
22	https://codeforces.com/problemset/problem/29/A
23	https://codeforces.com/problemset/problem/32/A
24	https://codeforces.com/problemset/problem/35/A

2 Контрольні запитання

1. У чому переваги цієї методики тестування ПЗ?
2. На якому етапі проектування ПЗ виконується проектування тестів при цьому підході?
3. Хто відповідає за розроблення тестів?
4. Хто проводить тестування?
5. Назвіть критерії завершення тестування ПЗ?
6. Хто відповідає за розроблення наборів даних для тестів?
7. Для чого треба розробити допоміжні модулі?

Лабораторна робота № 5. Тестування методами «білого ящика»

Мета. Вивчити методологію «білого ящика» для тестування функцій.

Для пропонованої функції ***F.XX, де XX*** – Ваш номер у списку з файла **варіант.txt,** необхідно передбачити контроль діапазону, типу даних і контроль помилок виконання. Після цього слід провести тестування виправленого модуля програми.

1 Хід роботи

1. Вивчити теоретичний матеріал (розділ № 5).
2. Сформувати набори тестів (вхідні дані й очікуваний результат), що задовольняють такі критерії покриття:
   - операторів;
   - рішень (розгалужень);
   - умов;
   - комбінованого покриття.
3. Провести випробування.
4. Порівняти очікувані результати з отриманими під час проведення тестування.
5. Занести в таблицю 1.1 тестові набори та знайдені помилки.
6. Оформити звіт. (Звіт про проблему в додатку Б).

Таблиця 1.1– Зразок оформлювання результатів тестів

№ п/п	Тестовий набір даних	Очікуваний результат	Отриманий результат

2 Контрольні запитання

1. У чому полягають переваги тестування методом «білого ящика»?
2. Які бувають критерії охоплення?
3. Як треба перевіряти умови?

4. Коли застосовують методи «білого ящика»?
5. Хто використовує методи «білого ящика»?

Лабораторна робота № 6. Тестування об'єктно-орієнтованого програмного забезпечення

Мета. Ознайомитися з методами тестування класів об'єктно-орієнтованого програмного забезпечення.

1 Хід роботи

1. Ретельно прочитати теоретичний матеріал (розділ № 6).
2. Уявити у вигляді UML-діаграми ієрархію класів ПЗ, що розробляли у домашньому завдані «Аналіз вимог до ПЗ».
3. Скласти набір тестових даних.
4. Провести вирівнювання ієрархії класів. Отриманий результат подати у вигляді діаграми.
5. Скласти таблицю умов тестування методів.
6. Запропонувати варіанти тестових прикладів у контексті всіх класів.

2 Контрольні запитання

1. Які переваги дають змогу отримати використання успадкування при об'єктно-орієнтованому програмування?
2. Яким способом виконується тестування успадкованих класів?
3. До яких переваг приводить вирівнювання ієрархії класів?
4. Для чого використовується драйвер для тестування класу?
5. Які існують основні принципи об'єктно-орієнтованого програмування і як вони впливають на етап тестування?

Лабораторна робота № 7. Тестування інтерфейсу «ПК-користувач»

Мета. Навчитися проектувати тестування інтерфейсу «ПК-користувач», аналізуючи можливість забезпечення компромісу між нижче переліченими факторами.

1. Функціональність.
2. Швидкість вивчення ПЗ новим користувачем.
3. Легкість запам'ятовування необхідних прийомів роботи.
4. Продуктивність ПЗ.
5. Імовірність виникнення помилок користувача при експлуатації ПЗ.
6. Задоволеність користувача ПЗ.

1 Хід роботи

1. Уважно прочитати теоретичний матеріал (розділ № 7) і вказати, які стилі взаємодії будуть реалізовані в ПЗ.

2. Визначити і записати всі вимоги до інтерфейсу «ПК-користувач», який буде реалізований, привести схему головної форми цього інтерфейсу.

3. Скласти таблицю рішень для тестування інтерфейсу «ПК-користувач», використовуючи номери типів помилок з таблиці 7.1 і певні умови кожного вимоги до інтерфейсу «ПК - користувач».

4. Указати послідовність тестування всіх опцій головної форми (і не тільки її) інтерфейсу.

5. Проаналізувати складену таблицю рішень на предмет неоднозначності.

6. Скласти звіт відповідно до порядку виконання роботи. Зробити висновки.

2 Контрольні запитання.

1. Скільки основних стилів взаємодії «ПК - користувач»?
2. Який взаємозв'язок між стилями взаємодії і типами додатків, в яких вони зазвичай використовуються?
3. Між якими факторами потрібно забезпечити компроміс?
4. Як враховують взаємодію між факторами?
5. Скільки існує типів помилок «ПК-користувач»?
6. Які завдання стоять перед розробниками інтерфейсу «ПК-користувач»?

Лабораторна робота № 8. Обчислення оцінки складності

Мета. Навчитися виконувати оцінку складності (статичну і динамічну) розробленого програмного продукту. Виміряти довжину і обсяг одного модуля.

1 Хід роботи

1. Ретельно прочитати теоретичний матеріал (розділ № 8).
2. Вибрати статичні складові складності програми, для якої визначили набори даних у лабораторній роботі № 4.
3. Вибрати динамічні складові складності для цієї ж програми.
4. Обчислити довжину цього модуля.
5. Обчислити обсяг цього модуля.
6. Порівняти реальний обсяг з обчисленим потенційним обсягом.
7. Оформити звіт.

2 Контрольні запитання

1. Як обчислюється обсяг програми?
2. Як обчислюється довжина програми?
3. Які метричні характеристики впливають на якість модуля?

4. Яку система метрик запропонував Холстед?
5. Що визначають статичні складові складності?
6. Який процес визначають динамічні складові складності?
7. На які види складності можна розділити динамічні складові?
8. Які існують способи оцінювання трудомісткості розроблення програмного забезпечення?
9. Які метричні характеристики залежать від алгоритму, реалізованого в модулі?

Лабораторна робота № 9. Обчислення рівня модуля

Мета. Навчитися обчислити рівень одного модуля.

1 Хід роботи

1. Ретельно прочитати теоретичний матеріал (розділ № 9).
2. Обчислити рівень програми, для якої визначили набори даних у лабораторній роботі № 4.
3. Визначити й описати інтелектуальний зміст одного модуля.
4. Оформити звіт.

.

2 Контрольні запитання

1. Від чого залежить інтелектуальний рівень програми?
2. Що входить у рівняння рівня програми?
3. Який взаємозв'язок між об'ємом і інтелектуальним рівнем програми?
4. Для чого треба визначати інтелектуальний рівень програми?

Лабораторна робота № 10. Тестування додатків для мобільних пристроїв

Мета. Навчитися виконувати верифікацію і тестування програмних додатків для мобільних пристроїв

1 Хід роботи

1. Ретельно прочитати теоретичний матеріал (розділ № 10).
2. Розробити для мобільного застосування плани системного та інтеграційного тестувань.
3. Обов'язково треба створити обґрунтовану кількість тестів для тестування всіх функцій. Кількість наборів даних для тестів треба оптимізувати, використовуючи методи «чорного ящика». Усі тести (з усіма наборами даних) необхідно зберігати в файлах.
4. Оформити звіт, що містить плани тестування, вказівки для послідовності виконання тестування, висновки.

2 Завдання для виконання тестування

2.1. Указати параметри пристрою.
2.2. Способи розроблення.

2.3. Визначення та тестування положення пристрою.
2.4. Контроль за використанням елементів управління.
2.5. Тестування мультизадачності.
2.6. Тестування комунікаційних процедур.
2.7. Тестування викликів Web-служб з мобільного пристрою.
2.8. Тестування умов і обсягу зберігання даних.
2.9. Указати рівень взаємодії додатків з зовнішніми ресурсами.
2.10. Тестування енергоспоживання.

3 Контрольні запитання

1. Які плани необхідно розробляти для тестування ПЗ для мобільних пристроїв?
2. Що необхідно вказати для виконання тестування ПЗ для мобільних пристроїв?
3. Як треба тестувати режим мультизадачності?
4. Що впливає на тестування енергоспоживання?
5. Як при тестуванні треба враховувати умови зберігання і обсяг даних?

Лабораторна робота № 11. Тестування WEB-додатка

Мета. Верифікація WEB-додатка.

1 Хід роботи

1. Ретельно прочитати теоретичний матеріал (розділ № 11).
2. Розробити план системного тестування WEB-додатка.
3. Розробити план інтеграційного тестування WEB-додатка.
4. Оформити звіт, що містить обидва плани. У кожному плані повинно бути обґрунтування і опис усіх тестів і наборів даних до цих тестів.

2 Контрольні запитання

1. Які основні категорії тестування WEB-вузлів?
2. Які особливості тестування функціональних можливостей WEB-додатків?
3. Які особливості тестування навігації по сторінках?
4. Які особливості тестування вмісту сторінок?
5. Які особливості тестування форм?
6. Які проблеми тестування WEB-додатків?
7. Які об'єкти містяться в плані системного тестування WEB-додатків?
8. Для чого розробляють план інтеграційного тестування?

Додаток Б

НАЗВА КОМПАНІЇ_____ КОНФІДЕНЦІЙНО_____

ЗВІТ ПРО ПРОБЛЕМУ №_____

ПРОГРАМА_____ ВИПУСК _____

ВЕРСІЯ_____

ТИП ЗВІТУ (1-6)_____ СТУПІНЬ ВАЖЛИВОСТІ (1-2)_____

1 - Помилка кодування
Якщо так, то які:
2 - Помилка проектування
3 - Пропозиція
4 - Розбіжність з документацією
5 - Взаємодія з апаратурою
6 - Питання

1 - Фатальна

2 - Серйозна

ДОДАТКИ (Д/Н)____

ПРОБЛЕМА_____
_____

ЧИ МОЖЕТЕ ВИ ЗРОБИТИ ПРОБЛЕМНУ СИТУАЦІЮ? (Д/Н) _____

ДОКЛАДНИЙ ОПИС ПРОБЛЕМИ І ЯК ЇЇ ВИРІШИТИ _____
_____
_____
_____

ПРОПОНОВАНІ ВИПРАВЛЕННЯ
(НЕОБОВ'ЯЗКОВО)_____
_____
_____

ЗВІТ НАДАНИЙ СПІВРОБІТНИКОМ _____ дата ___/ ___/ ___

НАСТУПНІ ГРАФИ ПРИЗНАЧЕНІ ЛИШЕ ДЛЯ РОЗРОБНИКІВ

ФУНКЦІОНАЛЬНА
ОБЛАСТЬ_____ВІДПОВІДАЛЬНИЙ_____

КОМЕНТАР_____
_____

Стан (1-2)_____ Пріоритет (1-3)__

1 - Відкрито 2 – Закрито 1- Низький 2 - Середній 3 - Високий

РЕЗОЛЮЦІЯ (1-9) _____ ВИПРАВЛЕНА ВЕРСІЯ _____

1 - Розглядається 4 - Відкладено 7 - Відкликано упорядником

2 – Виправлено 5 - Відповідає проекту 8 - Потрібна додаткова інформація

3 - Чи не відтворюється 6 - Не може бути виправлено 9 - Не згоден з пропозицією

РОЗГЛЯНУТО _____ дата ___/ ___/ ___

ПРОКОНТРОЛЮВАНО _____ дата ___/ ___/ ___

ВВАЖАТИ ВІДКЛАДЕНИМ (Д / Н)_____

Бібліографічний список

Боггс, У. UML и Rational Rose / У. Боггс, М. Боггс. – Москва : Лори, 2008. – 600 с.

Брауде, Э. Технология разработки программного обеспечения / Э. Брауде. – СПб. : Питер, 2004. – 655 с.

Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений на С++ : пер. с англ. / Г. Буч. – 2-е изд. – Москва : Изд-во «Бином», 1998. – 560 с.

Дастинг, Э. Автоматизированное тестирование программного обеспечения : пер. с англ. / Э. Дастинг, Д. Рэкшка, Д. Пол. – Москва : ЛОРИ, 2004. – 567 с.

Калянов, Г. Н. CASE. Структурный системный анализ (автоматизация и применение) / Г. Н. Калянов. – Москва : Лори, 1996. – 287 с.

Кватрани, Т. Rational Rose 2000 и UML. Визуальное моделирование : пер. с англ. (Сер. Объектно-ориентированные технологии в программировании) / Т. Кватрани. – Москва : ДМК Пресс, 2001.– 176 с.

Ларман, К. Применение UML 2.0 и шаблонов проектирования. Практическое руководство: пер. с англ. / К. Ларман. – 3-е изд. – Москва : ООО «И. Д. Вильямс», 2009. – 736 с.

Ларман, К. Применение UML и шаблонов проектирования: пер. с англ. / К. Ларман. – Москва : Изд. дом «Вильямс», 2001. – 496 с.

Маклаков, С. В. BPwin и ERwin. CASE – средства разработки информационных систем / С. В. Маклаков. – Москва : ДИАЛОГ-МИФИ, 2001. – 304 с.

Мацяшек, А. Анализ и проектирование информационных систем с помощью UML 2.0 : пер. с англ. / А. Мацяшек. – 3-е изд. – Москва : ООО «И. Д. Вильямс», 2009. – 816 с.

Мейер, Б. Объктно-ориентированное конструирование программных систем: пер. с англ. / Б. Мейер. – Москва : Изд. дом «Русская редакция», 2005. – 1232 с.

Microsoft Corporaition. Принципы проектирования и разработки программного обеспечения : учеб. курс 2004. – 544 с.

Якобсон, А. Унифицированный процесс разработки программного обеспечения / А. Якобсон, Г. Буч, Дж. Рамбо. – СПб. : Питер, MCSD : пер. с англ. – Москва : Изд. дом «Русская редакция», 2006. – 606 с.

Нормативная база программной инженерии в разработке систем с интенсивным использованием программного обеспечения: учеб. пособие / Б. М. Конорев, Л. Ф. Пудовкина, И. Б. Сироджа, О. Е. Федорович. – Харьков : Нац. аэрокосм. ун-т «Харьков. авиац. ин-т», 2001. – 162 с.

Орлов, С. А. Технология разработки программного обеспечения / С. А. Орлов. – СПб. : Питер, 2002. – 464 с.

"Основы Программной Инженерии (по SWEBOK)" - перевод SWEBOK 2004 с замечаниями и комментариями, подготовленный Сергеем Орликом при участии Юрия Булуя. http://swebok.sorlik.ru

Пудовкина, Л. Ф. Технология программирования и автоматизация проектирования программного обеспечения, метод. рекомендации для лаб. работ / Д. А. Бастеев, Л. Ф. Пудовкина. – Харьков : Нац. аэрокосм. ун-т «Харьков. авиац. ин-т», 2004. – 60 с.

Соммервилл, И. Инженерия программного обеспечения: пер. с англ. / И. Соммервилл.– 6-е изд. – Москва : Изд. дом «Вильямс», 2002.– 624 с.

Тассел, Д. В. Стиль, разработка, эффективность, отладка и испытание программ / Д. В. Тассел. – Москва : Мир, 1985. – 332 с.

Фаулер, М. Архитектура корпоративных программных приложений: пер. с англ. / М. Фаулер. – Москва : Вильямс, 2002. – 496 с.

Зміст

ВСТУП.....6
1 РОЗРОБЛЕННЯ ПЛАНУ СИСТЕМНОГО ТЕСТУВАННЯ.....7
1.1 Теоретичний матеріал 7
1.2 План системного тестування 9
2 РОЗРОБЛЕННЯ ПЛАНУ ІНТЕГРАЦІЙНОГО ТЕСТУВАННЯ.....10
2.1 Методи виконання інтеграційного тестування 10
2.2 Модульне тестування ПЗ 12
3 МЕТОДИ «ЧОРНОГО ЯЩИКА» 16
3.1 Метод функціональних діаграм 16
3.2 Класи еквівалентності даних.....20
3.3 Метод аналізу граничних значень вхідних даних 22
3.4 Метод припущення помилки.....22
4 РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ГЕНЕРАЦІЇ НАБОРУ ДАНИХ ДЛЯ ТЕСТУ 22
5 МЕТОДИ «БІЛОГО ЯЩИКА» 25
5.1 Теоретичний матеріал 25
5.2 Особливості методів «білого ящика» 25
5.3 Критерії охоплення.....26
6 ТЕСТУВАННЯ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 27
6.1 Метод успадкування.....27
6.2 Метод вирівнювання 28
7 ПРОЕКТУВАННЯ ТЕСТУВАННЯ ІНТЕРФЕЙСУ «ПЗ-КОРИСТУВАЧ» 30
7.1 Засоби взаємодій 30
7.2 Типи помилок 31
8 ОЦІНЮВАННЯ ЯКОСТІ ПРОГРАМ НА ОСНОВІ МЕТРИК ХОЛСТЕДА.....33
8.1 Теоретичний матеріал 33
8.2 Вимірні властивості алгоритмів 34
8.3 Довжина модуля.....35
8.4 Обсяг програми 36
8.5 Потенційний обсяг 36
9 ВИЗНАЧЕННЯ ІНТЕЛЕКТУАЛЬНОГО ЗМІСТУ ПРОГРАМИ.....37
9.1 Рівень програми 37
9.2 Висновок рівняння рівня програми 37
9.3 Визначення інтелектуального змісту програм.....38
10 ВЕРИФІКАЦІЯ І ТЕСТУВАННЯ ПРОГРАМНИХ ДОДАТКІВ.....39
ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ.....39
10.1 Етапи роботи. 39
10.2 Правила тестування 41
11 ТЕСТУВАННЯ WEB-ДОДАТКА.....45
11.1 Особливості тестування WEB-додатків 45
11.2 Плани тестування WEB-додатків 45

11.3 Виконання тестування WEB-додатка 47
ДОДАТОК А.....51
Лабораторна робота № 1. План системного тестування 51
Лабораторна робота № 2. Інтеграційне тестування 51
Лабораторна робота № 3. Тестування методами 52
«чорного ящика».....52
Лабораторна робота № 4. Генерація масиву даних для тесту 53
Лабораторна робота № 5. Тестування методами «білого ящика» 55
Лабораторна робота № 6. Тестування об'єктно-орієнтованого програмного забезпечення.....56
Мета. Ознайомитися з методами тестування класів об'єктно-орієнтованого програмного забезпечення.....56
Лабораторна робота № 7. Тестування інтерфейсу 56
«ПК-користувач».....56
Лабораторна робота № 8. Обчислення оцінки складності 57
Лабораторна робота № 9. Обчислення рівня модуля 58
Лабораторна робота № 10. Тестування додатків для мобільних пристроїв.....58
Лабораторна робота № 11. Тестування WEB-додатка 59
ДОДАТОК Б.....60
БІБЛІОГРАФІЧНИЙ СПИСОК 62
ЗМІСТ.....64

Навчальне видання

Постернакова Вероніка Альбертівна
Пудовкіна Лариса Федорівна
Туркін Ігор Борисович

ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ТЕСТУВАННЯ

Редактор С. П. Гевло

Зв. план, 2019
Підписано до видання 23.04.2019
Ум. Ддук. арк. 3,5 . Обл.-вид. арк. 4. Електронний ресурс

Видавець і виготовлювач
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»
61070, Харьков-70, ул. Чкалова, 17
http://www.khai.edu
Видавничий центр "ХАІ"
izdat@khai edu
61070, Харьков-70, ул. Чкалова,17

Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців, виготовлювачів і розповсюджувачів
видавничої продукції сер. ДК № 391 від 30.03.2001